ON LICENSING OF SOFTWARE ENGINEERS WORKING ON SAFETY-CRITICAL SOFTWARE

Final Report of an ACM Task Force

August, 2001

John Knight	*University of Virginia*
Nancy Leveson	*Massachusetts Institute of Technology*
Michael DeWalt	*Certification Services, Inc.*
Lynn Elliot	*Guidant, Inc.*
Cem Kaner	*Florida Institute of Technology*
Helen Nissenbaum	*Princeton University*

Last modification: October 12, 2001 2:19 pm

ACM TASK FORCE ON LICENSING OF SOFTWARE ENGINEERS WORKING ON SAFETY-CRITICAL SOFTWARE

1. BACKGROUND AND SCOPE OF REPORT

In 1998, the ACM was approached by the Texas Professional Engineers Licensing Board for help in defining performance criteria for software engineering licensing examinations to be administered in Texas. At the time, the ACM agreed to expand its on-going work with the IEEE Computer Society and the Software Engineering Coordinating Committee (SWECC) to include recommendations related to licensing examinations. This agreement was reached despite the fact that some members of the ACM Council had reservations about whether licensing software engineers was "in the best interests of the field of computing and the public."

In the summer of 1999 Barbara Simons created two blue-ribbon task forces: (1) to evaluate the IEEE/ACM Software Engineering Body of Knowledge Activities (SWEBOK) and (2) to determine ways in which ACM and the profession might improve the robustness and quality of safety-critical software and to evaluate the SWECC licensing activities in this context. This report describes the findings and recommendations of the latter committee.

In April 2000, the task force held a fact-finding meeting in Washington D.C. At that meeting, the task force had discussions with and got information from a group of invited experts. The presenters were: John Calvert, U.S. Nuclear Regulatory Commission; Paul Jones, U.S. Food and Drug Administration, Center for Devices and Radiological Health; Pat Natale, Executive Director National Society of Professional Engineers (NSPE); Arthur Schwartz, Deputy Executive Director & General Counsel, National Society of Professional Engineers; John Adams, National Council of Examiners for Engineering and Surveyors (NCEES); and James Moore, Mitre Corp, member of SWECC and SWEBOK.

The task force held a second meeting in April 2001 in Washington D.C. The second meeting focused on the ethical and legal aspects of the problem.

2. THE PROBLEM AND THE ORGANIZATION OF THE REPORT

Software is increasingly being used in systems that affect public safety, where safety is defined in the general sense as involving the possibility of unacceptable loss (including human life or injury, property damage, environmental pollution, societal disruption, large monetary loss, etc.). Such systems are referred to typically as *safety-critical* systems. While the first such software was carefully crafted by experts and was limited in its functionality, more extensive use and increased complexity of the software being used is leading to increasing numbers of software-related accidents and losses. Accidents almost always have multiple causal factors, but software has played a role in accidents involving, for example, medical devices, aircraft and other transportation systems, space vehicles, and defense systems. With the possibility of accidents being caused by defects in software, there is concern with protection of the public interest.

There have been suggestions by some government entities, by independent licensing bodies, and by individuals that software engineers should be licensed in order to protect the public interest.

Having a licensing requirement might not solve the problem, however, and, in fact, it could make it worse. The real issue is not licensing per se, it is to determine how best to protect the public without unduly affecting engineering progress, the economy, the professions, or individual rights.

Before starting down a path of required licensing or certification, we need to examine carefully exactly what licensing will mean and what its impact would be, and we need to consider all of the approaches that might contribute to safety. The various approaches that should be considered include mandatory licensing, certification, government regulation and oversight, voluntary product certification, insurance, standards, codes of practice, independent inspection (IV&V), ethical standards, liability and legal remedies, accreditation of educational institutions, and specification of a standard curriculum but without formal accreditation.

This report contains the *findings* and *recommendations* of the task force resulting from our internal discussions as well as interviews and meetings with experts on the various alternatives considered. Each finding is identified as such, and the statement of the finding is followed where appropriate by the rationale for the finding. Each recommendation is identified and constitutes a recommended action in response to the finding(s).

The reader is cautioned to keep the restricted goal of this report in mind while reading it. The subject of concern is the protection of the public interest for *safety-critical* systems. The primary area of study for the task force was the licensing using the Professional Engineer mechanism of software engineers engaged in the development of safety-critical software and possible alternative approaches, *not* for software engineering in general.

The report is organized as follows. In section 3 we discuss the issue of licensing. In section 4 we present several other approaches that could potentially make a positive contribution to the quality of safety-critical software. The role of the legal system is reviewed section 5 and the task force general conclusions and recommendations are presented in sections 6 and 7.

3. LICENSING AS A PROFESSIONAL ENGINEER

Licensing is employed in many branches of engineering using a common mechanism, the *Professional Engineer* or PE. One of the proposals that has been made regarding the licensing of software engineers is to utilize the PE mechanism to implement such licensing thereby aligning software engineering with the other engineering fields.

In this section we review the details of the PE mechanism and how it operates. We then present the task force's conclusions about the use of the PE mechanism for the licensing of software engineers.

3.1 What is a Professional Engineer?

A Professional Engineer (PE) is an individual who has been granted the right to use that title (i.e., has been licensed) based on a process that includes a four-year Accreditation Board for Engineering and Technology (ABET) Engineering Accreditation Committee (EAC) accredited university degree; an eight-hour examination on fundamentals of engineering taken usually in the

senior year of college; four years of acceptable experience; a second examination on principles of practice; and written recommendations from other professional engineers.

The practical implications of the PE license and the percentage of engineers licensed varies by engineering subdiscipline. It is important to note, however, that the PE license says "Engineer" and does not distinguish between subdisciplines. A PE can, however, be disciplined by the state (fined or lose their license) if they practice beyond their area of competence. Licensed professional engineers are accountable for their activities and assume legal liability.

Licensing is a state activity. The National Society of Professional Engineers (NSPE) provides a model law and lobbies legislatures to adopt licensing regulations. Every state currently has an engineering licensure law, covering such professions as surveyors, architects, geologists, and professional engineers. Who is an engineer is defined by "practice acts," i.e., defined by what the person does and not simply what they call themselves, although "title acts" also exist. It is illegal for somebody who is not licensed to use the title "engineer" or to practice an engineering profession in a state for which licensing is required. The rules for the licensing of PEs vary from state to state. Enforcement also varies. For example, the state of Oklahoma (and the model law) requires that faculty teaching engineering design classes must be licensed but this part of the Oklahoma law has not been enforced. However, it would apply potentially to any faculty member teaching a class in software engineering.

A professional engineer in professions for which licensing is required must be licensed in every state in which he or she practices. Mandatory licensing is usually required for those providing services directly to the public and those involved in the design of facilities, roads, transportation and construction where drawings etc. must be submitted to state agencies to be approved and which must be issued a seal by the state. There are sometimes exceptions to required licensing, most notably if the person works in industry (for a company) and does not provide direct services to the public or if the person is an employee of the federal government.

Licensing activities within each state are overseen by a state licensing board(s) whose members are usually appointed by the Governor. These boards act as a state agency and are supported by state appropriations. Most states charge licensing fees, ranging from $80-$200 per year. In many states, these fees go into the state's general appropriations budget. A person licensed in multiple states must pay the fees for each state in which they practice their profession. Some states allow granting licenses on the basis of being licensed in a different state whose requirements "meet or exceed" those of the state in question, but they must still pay in both states.

Current PE activities with respect to software are limited to the state of Texas and a few private advocates. There are many factors behind these efforts. One of these seems to be a legitimate concern among engineers (including those involved in PE licensing in states other than Texas) that the term engineer is being abused by activities by some computer companies (such as Novell and Microsoft) to certify people as "Certified Novell Engineers" or "Microsoft Certified Systems Engineers." In these cases, most people would agree that the use of the title "Engineer" is being abused and might be confusing to the public.

3.2 Findings and Recommendations on Licensing

Finding 1 on Licensing

- Licensing as PEs would be impractical for software engineering.

Rationale for Finding 1

1. As part of the PE licensing process, everyone must take an eight hour Fundamentals of Engineering examination (FE) composed of a morning general examination designed to match the ABET criteria for the first two years of an engineering degree and a discipline-specific afternoon examination that covers the material taught in the last two years of an engineering degree program.
   The morning general examination (that must be taken by everyone) covers:

   Chemistry: Acids and bases; equilibrium; equations; electrochemistry; inorganic chemistry; kinetics; metals and nonmetals; nomenclature; organic chemistry; oxidations and reduction; periodicity; states of matter; solutions; and stoichiometry.

   Computers: Algorithm flowchart; spreadsheets; pseudocode; and data transmission and storage.

   Dynamics: Force, mass, and acceleration; friction; impulse and momentum; kinematics; vibrations; and work and energy.

   Electrical Circuits: AC circuits; diode applications; DC circuits; electric and magnetic fields; capacitance and inductance; ideal transformers; Fourier and Laplace transforms; and operational amplifiers (ideal).

   Engineering Economics: Annual cost; breakeven point; benefit-cost analysis; future worth or value; present worth; and valuation and depreciation.

   Ethics: Relations with clients, peers, and the public.

   Fluid Mechanics: Flow measurement; fluid properties; fluid states; impulse and momentum; pipe and other internal flow; and similitude and dimensional analysis.

   Material Science/Structure of Matter: Atomic structure; crystallography; corrosion; diffusion; materials; binary phase diagrams; properties; and processing and testing.

   Mathematics: Analytic geometry; differential equations; differential calculus; difference equations; integral calculus; linear algebra; Laplace transforms; probability and statistics; roots of equations; and vector analysis.

   Mechanics of materials: Beams; bending; columns; combined stresses; shear; stress and strain; tension and compression; and torsion.

   Statics: 2-dimensional equilibrium; 3-dimensional equilibrium; centroid of area; concurrent force systems; friction; moment of inertia; and vector forces.

   Thermodynamics: 1st and 2nd law; availability-reversibility; cycles; energy, heat, and work; ideal gases; mixture of gases; phase changes; enthalpy, entropy, and free energy properties; and thermodynamic processes.

The afternoon, discipline-specific examination is offered in five disciplines and one general, non-specific discipline. The general, non-specific examination covers the same topics as in the morning examination but in more depth. A discipline-specific examination may be taken in chemical, civil, industrial, electrical, or mechanical engineering. Because of the requirements for there to be at least 100 ABET-accredited departments offering a degree in a subject to be included as a discipline-specific examination, software engineering could not be included for a long time, if ever, as an afternoon discipline-specific examination. We know of only one ABET-accredited school currently offering an undergraduate degree in software engineering.

Most PEs take the FE examination immediately upon graduation and the PE examination after four years of experience. It has been found that it is very difficult to pass the FE examination unless it is taken right before or after graduation from an engineering school.

In addition, the FE examination requirement for PE licensing is inappropriate for those receiving degrees from computer science departments that are not in Engineering Schools. Even computer science students in engineering schools are rarely trained in all the areas of engineering tested by the FE examination. Studying all those subjects would leave little time for studying computer science and the topics more relevant to software engineering.

After the practical experience period, an applicant for a PE must pass a second examination, this one on principles of engineering in a discipline-specific topic. The process of deciding what topics should be covered by the examination is long and rigorous. The current IEEE/ ACM SWEBOK effort and process would not suffice as it does not satisfy the requirements for a rigorous job analysis and a large and scientific demographic survey of randomly selected practitioners.

The National Council of Examiners for Engineers and Surveyors (NCEES) has been asked by the Texas Society of Professional Engineers to produce a software-engineering examination. They discussed the idea and decided they will do so only if software engineering met the criteria of engineering: that there is proof of need; and that there is an adequate number of people and ABET EAC (accredited) degree programs in software engineering to justify the effort and expense. These criteria are not met at the time of writing, and, therefore, they have no plans right now to create such an examination. If the criteria were met, the cost to produce an examination would be about $100,000 and would have to borne by some interested group. Without such an examination, software engineers would be required to take an in-depth examination in another engineering field to be licensed as a PE.

2. PE licensing requires graduation from an ABET accredited university department. Not all practicing software engineers graduate from departments that are candidates for ABET accreditation.

3. PEs are licensed in individual states and there are large variations among states in requirements, enforcement, etc. Large software development efforts often span multiple states, and software is usually sold in every state. Requiring software engineers to be licensed in each state in which a company for which they work has branches and offices in which they consult or the software they produce is used would be impractical. Some states have reciprocity agreements, but this is far from universal, and software engineers would still be required to go through the approval procedure and pay the licensing fees in each state.

4. The rate of technology change is slow in most fields and the NCEES process is very rigorous and fair. But the typical three year cycle to update a licensing examination will not be practical for many aspects of information technology, where changes are rapid and continual. The examination would most likely be out-of-date most of the time unless the NCEES examination update cycle could be reduced.
   In addition, licenses in most states are good for life. This raises the issue of whether a software engineer who got a degree in 1970 and has not updated his or her knowledge since that time is still qualified to create safety-critical software. About 15-20 states require some type of continuing education for licensed engineers, but this raises additional problems.
5. In response to legal challenges, the PE examinations are being changed to be all multiple choice to ensure objective grading. There must be one answer that is correct, and the rest must be demonstrably incorrect. It is unlikely that any reasonable test of software engineering skill for safety-critical systems can be put into this format.
6. The breadth of people who are somehow involved in the production of safety-critical software would make licensing all of them impractical and not particularly helpful. Who would be included: requirements writers? designers? coders? software librarians? quality assurance and testers? reviewers of the software? those who select the computer hardware, operating systems, and provide the system software (such as schedulers and operating systems for real-time systems)? managers?
   Much of current embedded software is created not by software engineers but by other engineers who use graphical programming languages and systems such as MatrixX. This raises the question of whether they are practicing software engineering. By the standards of the PE process, they would not be required to be licensed or tested in any field but their major engineering discipline (such as mechanical engineering). Similarly, the issue has to be resolved for a person who uses a spreadsheet, for example, or configures an SAP system.

Finding 2 on Licensing

- Licensing software engineers as PEs would have no or little effect on the safety of the software being produced.

Rationale for Finding 2

1. Licenses are usually only required for those engineers (and others such as lawyers, electricians, day care workers, etc.) who provide services directly to the public. Thus consultants and teachers are usually covered but employees who work for companies that produce safety-critical products (and those working for the government) are, in almost all states, exempted from licensing requirements. Therefore, licensing is unlikely to have any effect on safety as virtually all safety-critical systems are built by large teams of people who work for a company. There are few instances where safety-critical software is produced by an individual and sold directly to the public. Most software engineers would not fall under any licensing regulations and would not be required to be licensed nor find much benefit in voluntary licensing.
2. The current PE examination is aimed at a pass rate determined by “minimal competence”. The level of competence necessary to pass the current examination is unlikely to affect the

quality of safety-critical software. Past minimal competency examinations in computer science areas, such as the DPMA, have been widely ignored and are essentially irrelevant.

3. Few engineers actually are licensed as PEs, with the majority being in civil engineering (about 40% of civil engineers) and the second largest being mechanical engineers. The lowest percentage is electrical engineers (about 10%). There is currently a decreasing percentage of mechanical and electrical engineers going through the licensing process while the percentage of civil engineers is holding steady.
4. The PE process licenses everyone as an "engineer"—it is up to the individual to determine whether they are qualified to practice a particular subdiscipline. Therefore, requiring a software engineer who builds or approves safety-critical projects to be a licensed PE is unlikely to solve any problems related to engineers who have limited software expertise.

Recommendation on Licensing

- No attempt should be made to license software engineers engaged in the development of safety-critical software using the existing PE mechanism.

4. OTHER APPROACHES

Although a large amount of attention has been directed at the possibility of licensing software engineers, this is not the only possible approach to improving software engineering practice and tackling safety issues. The task force discussed eight other approaches to improving software engineering practice: (1) the development and use of a statement of ethical responsibility and of codes of professional conduct; (2) the specification of a standard software-engineering curriculum or body of knowledge; (3) revision of education strategies; (4) increasing oversight and regulation by the government; (5) the development of standards; (6) the development of codes of practice; (7) increasing the use independent inspection; and (8) actions by the insurance industry and voluntary product certification.

To greater or lesser extent each of these approaches is in use in software development. For example, independent inspection in some form is quite common, and government oversight by the Federal Aviation Administration is an integral part of the development of all critical avionics systems in commercial air transports. The issue that arises is the utility of these approaches, singly and in combination, in protecting the public interest with regard to safety-critical software.

Each of these approaches is discussed in this section. In all cases specific findings about the approach are presented and, in some cases, recommendations are presented.

4.1 Ethical Responsibility and Codes of Professional Conduct

Many professional organizations have codes of conduct or codes of responsibility to express what it means for a practitioner to be in good standing, prescribing a great many aspects of what it means to be a good "X", from behavior to attitudes to expertise. For purposes of this report, it is helpful to place these prescriptions into three categories:

- *Moral duties—Ethics*
  Some codes list a set of professional duties that are derived from general moral principles, such as refraining from harm, violating privacy, being dishonest, etc. Typically, these are

expressed in fairly abstract terms and need considerable interpretation to say what they mean in terms of specific actions or —more relevant to this report—standards of care.

- *Commitments to social values—Values*
  Codes may express a professional organization's commitments to certain social values. These might include justice and fairness, trust, social welfare, and equality. For software engineering of safety-critical systems, it will be particularly important for designers and engineers to be sensitive to societal concerns about and acceptance of risk versus societal interest in innovation and experimentation that may need to be traded off against risk.
- *Commitments to professional excellence—Competence*
  Practitioners need to perform at the best level they can. Codes may draw explicit links between practical excellence (knowledge, expertise, know-how) and the moral obligations of practitioners.

The question that the profession has to ask is: "How can the most effective use of ethical responsibility and codes of professional conduct be made to promote the highest possible standards of care and quality in software engineering?" This is an extremely difficult question to answer. In practice, it has been the subject of intense study by ethicists over a long period of time in many branches of engineering, and much can be learned from this experience that applies here.

For the most part, statements of ethical responsibility and codes of professional conduct serve to convey to practitioners and society at large the understanding that the profession has of what it means to be a practitioner in good standing. However, a good code, staunchly backed by an enforcement agent such as a professional society, *also* serves to empower individual members, who are committed to good practice, against outside pressures (typically, employers) to compromise these standards.

Findings on Ethical Responsibility and Codes of Professional Conduct

- A properly formulated code of professional conduct together with an effective enforcement mechanism is an important component of a comprehensive approach to protecting the public interest in regard to safety-critical software.
- Formal education in engineering ethics should be included in the education received by properly trained software engineers. This is an ABET requirement.
- In order for codes of professional conduct to be effective, professional societies need to invest effort in ensuring their codes are properly designed at the outset and properly maintained over time. Designing a code of professional conduct in an engineering discipline is a substantial challenge that has been studied in depth (see, for example, the work of John Ladd, Professor Emeritus of Philosophy at Brown University).
  Codes need to be maintained as "live" documents. This means having continuous reviews at reasonable intervals to accommodate progress in the field.

Recommendations on Ethical Responsibility and Codes of Professional Conduct

- The ACM should review its present "code of ethics" with experts in the field and make appropriate amendments.

- The ACM (and the other professional societies) should consider developing suitable classroom materials for teaching ethics in the field of software engineering, publishing regular columns in appropriate journals that discuss ethical issues, and working with ABET to ensure proper education in software engineering ethics is provided to computer science and computer engineering students.
- The ACM (and the other professional societies) should endorse their codes publicly. This might include: (1) placing links to the code in a prominent place on their websites; (2) including the code on membership and renewal forms; (3) requiring applicants to sign indicating acceptance of the code; and (4) making explicit statements to the effect that the professional society holds members accountable to the code. In this regard, it is important that a code has and is seen to have some "teeth", i.e., there will be appropriate penalties for violations of the code.

4.2 Specification and Application of a Software Engineering Body of Knowledge

An extensive project has been undertaken to prepare what is referred to as a software engineering body of knowledge (SWEBOK). The intent is to codify the material considered essential for the successful practice of software engineering. The anticipated/apparent application was to support the mechanics of licensing. Another ACM committee evaluated SWEBOK in depth and we do not repeat their conclusions here. Instead, we concentrate on issues related to safety.

Findings on Software Engineering Body of Knowledge

- There is no generally agreed comprehensive body of knowledge for software engineering of safety-critical systems at this time. Different systems require different techniques, and all systems require the application of a wide range of principles and techniques in order to produce a system with the requisite safety properties for a given application.
- The documentation of requisite software-engineering knowledge, even if it could be done, is not sufficient to make a significant contribution to improving safety because such documentation cannot convey experience with, judgement about, and applicability of the knowledge.

Rationale for Findings

1. Safety-critical software and systems have special needs and knowledge that are excluded from the IEEE SWEBOK effort (for example, real-time systems are excluded). Only universally required and applicable knowledge is being included. Thus it will have limited relevance to safety-critical systems, might exclude important knowledge related to competence to build these systems, and might imply that having the knowledge that is included will make a person qualified to build safety-critical systems.
2. Relatively little scientific evaluation of software engineering techniques has been done and it will be difficult to get consensus on what should be included and what should be excluded.
3. Few textbooks or programs exist that provide adequate instruction on building safety-critical, real-time software. Standard software-engineering textbooks and classes do not provide adequate instruction in these special aspects of software engineering and thus do not prepare students to work on such systems. This is particularly significant for the specialized

knowledge associated with application software where knowledge of important software-engineering concepts is required along with knowledge of the application domain.

Recommendation on Software Engineering Body of Knowledge

- The creation of a single software body of knowledge would have very little or no effect on safety and, for safety-critical software, should not be pursued. Instead, attention should be focused on the general education system (see section 4.3), particularly with respect to the knowledge required to engineer real-time, safety-critical software.

4.3 Improved University Education

The primary sources of skilled engineers in all branches of engineering are universities and colleges offering four-year degree programs. In most classical fields of engineering the universities have defined degree programs that provide adequate training. In software engineering this is far from the case and the task force made a number of findings and developed a number of conclusions relating to software engineering education in universities and colleges.

The issue of accreditation was also discussed and the situation with accreditation was found to be rather less clear than with the general topic of content and practice in software engineering education.

Findings on University Education

- An appropriate university education conveys to the engineer a wide variety of knowledge about many engineering principles and techniques, experience in their use, the ability to exercise judgement in their application, and an awareness of his or her own competence to undertake various engineering tasks.
- The information needed by software engineers to build safety-critical software is not included in the curricula of many university departments. Examples of topics that are not included typically are real-time software, software testing, safety analysis, fault-tolerant design, simulation, and design of human-computer interaction.
- Students intending to practice software engineering for safety-critical applications do not learn enough about computer applications and the other engineering disciplines. Knowledge of application areas is crucial in safety-critical software engineering since much of what is required to develop such software is actually system information. Similarly, students in other engineering disciplines do not have sufficient understanding of software engineering and, as a result, do not comprehend the difficulties of building software properly, the limitations of software in general, or appropriate techniques for facilitating safety in software intensive systems.
- There is considerable variation in the departments that teach software engineering at different universities. In some cases it is taught in computer science departments, in others in computer engineering departments (and sometimes both), and in still others in related departments such as mathematics. There is beginning to be a trend in which specific engineering disciplines such as mechanical, civil, and aerospace engineering teach their own students about software engineering and even award degrees in it.

- Different knowledge will be required for software developers working in different industries (e.g., nuclear power vs. financial systems) and different types of software will be required (e.g., real-time process control vs. information or database systems). Thus it is unlikely that one curriculum will be found that will teach what is required to produce appropriate software in these various circumstances. Multiple curricula, geared to particular types of systems, may be more practical.

Recommendations on University Education

- The ACM (and the other professional societies) should define model curricula for a variety of educational goals and industries. This effort should include input from a broad section of the community and reflect the essential variation and the diversity of opinion on this topic rather than produce a single "correct" curriculum.
- The ACM (and the other professional societies) should work with ABET in the accreditation of new software engineering programs in engineering schools in order to ensure that all relevant accreditation issues are considered.

4.4 Government Oversight And Regulation

There is a large variation in the approaches that are taken by the various government agencies tasked with regulating particular industries. Much of this variation is tied up with historical, political, and cultural issues rather than technological factors. In general, government oversight and regulation has been effective in protecting the public safety for those industries that are regulated. However, for political and historical reasons, many countries and industries do not accept government regulation or oversight or do so only to varying degrees.

The present approaches to regulating products and systems that include software (such as are employed by the FAA and the FDA) vary from approval only at the overall system design level to detailed inspection of the process used to develop the software and/or the resulting software product itself.

Findings on Government Oversight

- The degree of responsibility for safety varies among government agencies. In addition, although government regulation can be useful, any form of independent inspection and evaluation of software is necessarily limited in its effectiveness, and therefore responsibility for safety must ultimately lie in the hands of those designing and manufacturing the software and products.
- When accidents or anomalies occur it is important to learn as much as possible from the event. Better reporting and documentation of what went wrong would be extremely useful in improving government oversight procedures (where they apply) and the practice of software engineering in general. However, liability concerns often limit the information companies are willing to make public about software errors or, more generally, design errors in their products.
- Because of the fundamental difference between safety and reliability, the most effective oversight will involve special safety-related software engineering practices and will not simply focus on software reliability.

Recommendation on Government Oversight

- The ACM (and the other professional societies) should consider working with government agencies to establish an anonymous reporting system operated by an independent organization (perhaps similar to the FAA/NASA Air Safety Reporting System, the Marine Safety Reporting System, or the Nautical Institute International Marine Accident Reporting Scheme) to collect and evaluate data on "what went wrong" in the software engineering of safety-critical systems. Great care would have to be exercised to ensure accuracy and confidentiality of the data. If this data were analyzed, structured and appropriately presented, it could have a substantial impact on the prevention of future accidents.

4.5 Standards

A large number of software standards, well into the hundreds, have been produced by a variety of organizations on nearly every subtopic within software engineering, including safety-critical software. Organizations such as the International Standards Organization (ISO), the American National Standards Organization (ANSI), and the IEEE each maintain dozens of standards that relate in some way to software. In some cases standards are common where one standards organization adopts a standard developed by another without change.

In the software engineering field, standards have been produced for software development processes, specific software engineering techniques, terminology, and software project management among others. There are even standards about standards, e.g., *IEEE Standard Taxonomy for Software Engineering Standards*.

An important factor in considering the potential role of standards in safety-critical system is the fact that standards are never tested before they are published. Thus, although they might have been subjected to public review, they will not have been used and evaluated in practice. The result is that standards are not necessarily complete or consistent, and there is no assurance that application of a given standard will yield a specific desired goal.

Findings on Standards

- Standards help to define terminology, standard interfaces, etc., which benefit communication and interoperability. The current standards process does create standards that codify current practice to some degree.
- The process of creating a standard is almost always lengthy since it is usually undertaken by a committee. Membership of standards committees are usually volunteers with no requirements on qualifications, i.e., the process is open to anyone with the time and means to participate.
- There has been no real scientific evaluation of software engineering standards or most of the processes and techniques described in these standards, and there exists little scientific evidence within the software engineering community on which to base standards.
- Many of the processes for developing standards are long and require large commitments of time. Most companies are unwilling to allow their most valuable employees to participate with the result that existing standards display great variance in quality. In fact, some

international standards for safety-critical software have been criticized for promoting unsafe practices.

- The length of the standards process means that standards tend to be out of date, behind the state of the art, and even behind the state of the practice.
- There appear to be too many standards. There is difficulty in deciding which ones should be used and under what conditions. The lack of evaluation makes selection of appropriate practices and standards problematic.
- Because standards are consensus based, they tend toward minimal requirements. Some fraction of the community must agree and this tends to remove controversial topics. The more stakeholders involved in establishing a standard, the more difficult it becomes to include controversial or non-minimal requirements.
- Technically detailed, valid, and comprehensive standards are unlikely to apply to all companies and projects.
- Some companies depend on the IEEE and other professional society standards to act as an authority on the best practices to apply and, in some cases, as protection from malpractice and legal liability.

Recommendation on Standards

- The ACM should support the establishment of a process for evaluating the proliferating standards pertaining to safety-critical software engineering.
- The ACM should provide guidance to its members in selecting appropriate standards for use in safety-critical systems.

4.6 Codes of Practice

A code in this context is a system of principles or rules that practitioners are required to follow in order to ensure that products have certain properties. Codes of practice codify the requirements to do a job, e.g., building and electrical codes. National Electrical Code, for example, states the following requirement about wiring:

> *Conductors shall be racked to provide ready and safe access in underground and subsurface enclosures into which persons enter for installation and maintenance.*

Codes are a very effective way to pass down experience derived from accidents and serious losses. They differ from standards in that they specify required properties of the product, i.e., the results of a process rather than the process itself. Standards, in contrast, are more general and may specify product or process characteristics.

Findings on Codes of Practice

- In the trades, codes of practice are based on real science (e.g., Ohm's law) and experience (particularly negative experience). In software engineering they have the potential to be very powerful, but need to be based on science or adequate experience. Otherwise, they could have a negative impact by imposing practices that are ineffective or limit choices (e.g., all

safety-critical software must be written in Visual BASIC or must be object oriented or all software products must be documented using UML).

- Unlike the trades, any code of practice for software engineering would likely have narrow applicability and would need to be tailored to specific types of applications (as is true for existing codes in general engineering fields).

Recommendations on Codes of Practice

- The routine use of codes of practice in the development of safety-critical software is inappropriate because there is insufficient scientific basis for the creation of a comprehensive code.
- Research needs to be undertaken to establish sound scientific principles upon which an appropriate set of codes of practice for safety-critical software engineering could be built.

4.7 Independent Inspection

Independent inspection, often labeled Independent Verification and Validation (IV&V) is literally a process of review and testing of a product that is carried out by an organization different from the one that developed it. In some cases, the technique is required by government agencies for safety-critical software and in others it has been adopted voluntarily.

Independent inspection contributes more than a mere check of certain properties of a product. The fact that it is *independent* means that the inspection is carried out by individuals with different perspectives from the developers. This promotes the detection of defects in many areas. There is also a tendency for developers to be more careful with the creation of a software system if they know that the system will eventually be inspected by an independent group.

Companies exist that perform various types of independent inspection, and government agencies have sometimes created entities without their agency to perform this task on their projects.

Finding on Independent Inspection

- Independent software verification and validation, when practiced well, has been very effective in ensuring high-quality, safety-critical systems for those industries and agencies that have required them.
- Like anything else, the quality and effectiveness of software IV&V varies with the actual procedures performed and the experience and skill of those performing them.

Recommendations on Independent Inspection

- Procedures for the most effective use of independent inspection should be developed and evaluated.
- Independent inspection should be vigorously promoted as an accepted and desirable practice in the development of safety-critical software.

4.8 The Insurance Industry and Voluntary Product Certification

Insurance companies indemnify policyholders against unacceptable accidental loss, and in order to reduce losses the insurance industry has been proactive in promoting voluntary product

certification. The insurance industry and voluntary certification organizations, such as Underwriter's Laboratories (UL), have been extremely successful in reducing certain types of accidents, particularly those related to electrical and fire safety. UL has branched out recently to include not only electrical devices but programmable systems and a company's quality process. UL provides testing to assess conformity to some type of standard. They have created and are selling their own UL standard for safety-critical software and, for a fee, will test software-controlled products.

At least one large insurance company, Lloyd's of London (actually a set of underwriting syndicates rather than an insurance company) has established a group to evaluate risk in software-controlled systems. Insurance companies can potentially have a tremendous influence on engineering practices by basing insurance coverage and costs on the use of specific practices or on their own or other's conformity testing.

Finding on The Insurance Industry and Voluntary Product Certification

- Insurance companies could be very powerful in exerting pressure on companies to use particular software engineering practices if they decided to do so. A few large losses could move them in this direction. Whether this is a positive or negative development will depend on what practices they encourage or require.
- While certification of products containing software might be feasible, voluntary certification of software by itself could be misleading and dangerous as the safety of software is totally dependent on the design of the larger system in which it is contained. The same software that is safe within one product may be unsafe within a slightly different one. That is, safety is a property of the software combined with the design of the larger system in which the software is embedded and not a property of the software itself.
- The new UL standard for safety-critical software is inherently no different than the standards developed by professional societies although the process for developing the standard differs and is less open to public participation. Such standards have the same limitations as other standards and, if not adequate, have the potential to cause great danger to the public by companies and individuals relying on them to protect themselves and using them (as well as voluntary product certifications) as a defense in legal proceedings.
- Once again, the lack of scientific foundation for selecting among software engineering practices will limit the potential effectiveness of any attempts to ensure safety by external pressures to adopt certain processes or practices.

Recommendations on The Insurance Industry and Voluntary Product Certification

- The ACM should encourage the insurance industry to work with experts in safety-critical software engineering to promote established safety techniques by requiring their use in order to obtain or reduce the cost of insurance coverage.
- Certification of the safety of software separate from the system in which it will operate should not be permitted.

4.9 Voluntary Certification of Software Engineers

Voluntary certification, as a separate issue from state-mandated licensing, was not discussed extensively by the task force. This section is included to point out some of the similarities and differences.

The notion of *licensing* is to have some authority grant permission to an individual to engage in an activity that is otherwise unlawful. In typical professional situations, a license is only granted to an individual who has satisfied specific educational, knowledge, and/or and experience requirements. Note that any licensing that involves an exam cannot determine *competence* but simply a minimum level of knowledge.

Similarly, *certification* assures that an individual meets a minimum set of requirements. Like licensing, the actual requirements may involve particular education degree achievements, a minimum level of knowledge as determined by an examination, and/or a particular amount or type of work experience. Certification may cover a very large domain, such as all of software engineering; more limited but still general domains such as safety-critical, real-time software engineering or web site design; or may involve knowledge specific to a particular product or manufacturer such as certification by Microsoft or Novell on their own products.

Licensing and certification differ primarily in the permission to act. Licensing is mandatory and is a state or federal activity (usually state) while certification is voluntary. The two activities are sufficiently similar that almost all of the issues and concerns mentioned in this report about licensing apply equally to certification and we do not repeat them here. One difference is that because of the voluntary nature of certification, it can only be successful if it is accepted or supported by a majority of practitioners and employers. Past attempts to provide certification of software professionals have failed because the community ignored them.

Finding on Voluntary Certification

- Some of the arguments in favor of certification (and licensing) have confused the concept with gaining respect for software engineering and with being a profession. Neither is achieved by nor dependent on certification (or licensing).
- Certification (and licensing) do not assure competence, but only satisfaction of a minimum level of usually knowledge-based requirements.
- Certification of individuals by companies to assure knowledge about their particular products is a useful activity provided the associated title is not misleading.
- Before engaging in creating a certification process for software engineers, professional societies should ensure that such a process would be acceptable to and supported by those practicing the profession. Without this support, such certification would at best be ignored and at worst could create a backlash and divisions within the software engineering community.
- While more general certification has obvious benefits, it also has some serious dangers. It is important that attempts by the professional societies to certify software engineers in general or even more limited areas be considered very carefully before diving into something that could have important negative impacts and side effects.

5. THE ROLE OF THE LEGAL SYSTEM

The legal system does not provide an explicit approach to achieve the goal of protecting the public interest. Rather, it works with all the other approaches to provide a meaning to many of the concepts embodied in the other approaches and to provide a path for action subsequent to a failure within one of the other mechanisms.

With this in mind, the discussion of the legal system is placed in a separate section after all of the approaches considered. Reference is made in this section to many concepts and terms that are discussed in sections 3 and 4, and so this section supplements those in several important ways.

Background

There is no tort of computer malpractice today. That is, courts have consistently rejected attempts to sue software engineers and computer and software manufacturers liable for computer-related malpractice. In explaining their decisions, courts note that malpractice (professional negligence) involves harm caused by the failure of a person to perform within the standards of her profession. Software engineering is not a licensed profession, they say, and therefore software engineers are not subject to malpractice suits. If software engineers are licensed, they will be subject to malpractice lawsuits, and they will, most likely, need to carry malpractice insurance which could be very expensive (e.g., premiums in some professions are as high as $50,000 per year even for an individual with a good claims history).

The process of determining that an act constitutes malpractice is only partially driven by engineers. In a typical lawsuit for malpractice, an injured or economically harmed person sues the engineer, often as part of a broader lawsuit. The person will bring evidence that the engineer acted in ways, or made decisions, that were not up to the professional standards of software engineering. This evidence will be evaluated by lawyers, liability insurance company staff and lawyers, jurors, and judges. None of these people are engineers. If juries find that certain conduct is negligent, malpractice insurers will probably advise their insureds (engineers) against engaging in that conduct and may well provide incentives, in the form of lower insurance rates, for engineers who adopt recommended practices or who do not engage in counter-recommended practices. Over time, the determination of what constitute good engineering practices may be driven more by the courts and insurance companies than by engineers.

Over the past 30 years, American juries were required to evaluate a wide range of engineering design issues in products liability suits. This caused enormous problems and led to major changes in the law (resulting in the adoption of a new Restatement of Products Liability that completely redefined the standards for holding a manufacturer accountable for a product's design defect.) Unless the professional standards for software engineering are very clear, throwing determination of what constitutes good engineering practice to the courts is a reckless practice.

Standards for software engineering, including the IEEE standards and SWEBOK and those being produced by other professional societies and independent entities such as UL, are not tightly linked to empirical evidence and other scientific research. The existence of these standards, in the context of a licensed profession, puts professionals at risk. What should engineers do when they believe that a given practice is the best under the circumstances, but that practice is either recommended against in SWEBOK or a standard or a different practice is recommended? If they

follow their best judgment and a problem later arises, it will be easier to prove that they committed malpractice (whether or not their decision was actually the best one under the circumstances and not malpractice at all.) Should they instead follow a poor standard against their better judgment, in order to limit their malpractice litigation risk? Because the standards are based on consensus of the standards-writers, and are not tightly linked to science, it is entirely possible that the engineers' judgment will be much better than the practice that would follow from a given standard.

The legal system depends upon the ability to make appropriate judgements in response to questions. Yet, as noted earlier in this report, it is unclear who would have to be licensed and what practices would constitute the unlicensed practice of software engineering in the present proposals. Different speakers and authors have led us to different answers to this question, and we know of no authoritative source that will give us unambiguous guidance at this time.

Findings on Legal System

- The current efforts to pass laws such as the Uniform Computer Information Transactions Act (UCITA) to protect software manufacturers from being held responsible for the damage done by their product, if successful, are likely to have a much greater negative impact on safety than any efforts to improve safety as described in the other sections of this report.
- The large number of standards and little consensus in the field about acceptable practice and educational curriculums makes it very difficult for practitioners to determine under which standards or for which practices they will be held accountable.
- Warranties are effective in maintaining minimum standards in companies that are required to or voluntarily provide such warranties on their products. In the software industries, broad disclaimers are common. UCITA-like efforts are moving in the direction of providing even less liability by the manufacturers of software and legitimizing blanket disclaimers.

Recommendation on Legal System

- ACM should adopt the following principles with respect to the way the legal system deals with software:
    - Attempts to pass blanket limitations on the liability of software developers should not be supported.
    - Clauses in software licenses that restrict reverse engineering should be unenforceable against reverse engineering done for the purpose of discovery or proof of safety or security related problems.
    - Clauses in software license that restrict publication of benchmark studies or other product reviews should be unenforceable against revelation of safety or security related problems.
    - The maker and vendor of a software product should be required to reveal all of the defects that they know about in that product, and to document them in a way that a typical member of the market for that product can be reasonably expected to understand.
    - The customer of a software product, including software embedded in a device, should be able to see the terms of the software license before incurring an obligation to pay for the product.

- If a software product, including software embedded in a device, is the cause of an injury to a person or to damage to physical property, the terms in the license should not be enforced if they (i) select the law of a state that is neither the state in which the injured party lives or the state in which the injury took place, (ii) require the injured person to travel to another state, or (iii) limit damages below those that would normally available in a products liability suit.

6. General Findings and Conclusions

The committee agreed that an effective approach to ensuring safety in software-intensive systems will require a system's approach and cannot be achieved by dealing with software in isolation from the products and systems in which it operates.

The committee agreed that an effective approach to ensuring safety in software-intensive systems will require a carefully considered combination of the techniques discussed in this report (and perhaps others), and that no single approach is likely to be effective. Each industry will need to determine an appropriate mix of approaches that work together to solve their particular problems and fit within the cultural context of that industry. There are no simple and universal fixes that will solve the problem of ensuring public safety.

7. Recommendations

For the purpose of protecting the public interest in the context of safety-critical systems, many specific recommendations are made throughout the body of this report. They are repeated here to provide ready access:

- No attempt should be made to license software engineers engaged in the development of safety-critical software using the existing PE mechanism.
- The ACM should review its present "code of ethics" with experts in the field and make appropriate amendments.
- The ACM (and the other professional societies) should consider developing suitable classroom materials for teaching ethics in the field of software engineering, publishing regular columns in appropriate journals that discuss ethical issues, and working with ABET to ensure proper education in software engineering ethics is provided to computer science and computer engineering students.
- The ACM (and the other professional societies) should endorse their codes publicly. This might include: (1) placing links to the code in a prominent place on their websites; (2) including the code on membership and renewal forms; (3) requiring applicants to sign indicating acceptance of the code; and (4) making explicit statements to the effect that the professional society holds members accountable to the code. In this regard, it is important that a code has and is seen to have some "teeth", i.e., there will be appropriate penalties for violations of the code.
- The creation of a single software body of knowledge would have very little or no effect on safety and, for safety-critical software, should not be pursued. Instead, attention should be

focused on the general education system (see section 4.3), particularly with respect to the knowledge required to engineer real-time, safety-critical software.

- The ACM (and the other professional societies) should define model curricula for a variety of educational goals and industries. This effort should include input from a broad section of the community and reflect the essential variation and the diversity of opinion on this topic rather than produce a single "correct" curriculum.
- The ACM (and the other professional societies) should work with ABET in the accreditation of new software engineering programs in engineering schools in order to ensure that all relevant accreditation issues are considered.
- The ACM (and the other professional societies) should consider working with government agencies to establish an anonymous reporting system operated by an independent organization (perhaps similar to the FAA/NASA Air Safety Reporting System, the Marine Safety Reporting System, or the Nautical Institute International Marine Accident Reporting Scheme) to collect and evaluate data on "what went wrong" in the software engineering of safety-critical systems. Great care would have to be exercised to ensure accuracy and confidentiality of the data. If this data were analyzed, structured and appropriately presented, it could have a substantial impact on the prevention of future accidents.
- The ACM should support the establishment of a process for evaluating the proliferating standards pertaining to safety-critical software engineering.
- The ACM should provide guidance to its members in selecting appropriate standards for use in safety-critical systems.
- The ACM should encourage the insurance industry to work with experts in safety-critical software engineering to promote established safety techniques by requiring their use in order to obtain or reduce the cost of insurance coverage.
- Certification of the safety of software separate farm the system in which it will operate should not be permitted.
- ACM should adopt the following principles with respect to the way the legal system deals with software:
    - Attempts to pass blanket limitations on the liability of software developers should not be supported.
    - Clauses in software licenses that restrict reverse engineering should be unenforceable against reverse engineering done for the purpose of discovery or proof of safety or security related problems.
    - Clauses in software license that restrict publication of benchmark studies or other product reviews should be unenforceable against revelation of safety or security related problems.
    - The maker and vendor of a software product should be required to reveal all of the defects that they know about in that product, and to document them in a way that a typical member of the market for that product can be reasonably expected to understand.
    - The customer of a software product, including software embedded in a device, should be able to see the terms of the software license before incurring an obligation to pay for the product.

- If a software product, including software embedded in a device, is the cause of an injury to a person or to damage to physical property, the terms in the license should not be enforced if they (i) select the law of a state that is neither the state in which the injured party lives or the state in which the injury took place, (ii) require the injured person to travel to another state, or (iii) limit damages below those that would normally available in a products liability suit.

In addition to these specific recommendations, the task force makes the following general recommendation to the ACM:

- The ACM should work with its members, other professional societies including the IEEE Computer Society and the societies for related engineering disciplines such as the American Institute of Aeronautics and Astronautics to discuss, develop and promote a comprehensive set of actions to protect the public interest in regard to safety-critical software engineering.

