Virtually everybody in the industrialized world is affected by critical systems using software every day -- from your automobile ignition to ground and air traffic control as well as medical systems.

This talk will describe the critical engineering ethics issues and their relationship to the profession. This will lead into practical approaches for the implementation of software-intensive systems that are safe and minimize future liability.
The scope does not include System Safety Engineering except as it interfaces with the development of safety-critical software.

The bibliography particularly recommends certain sections of Prof. Nancy Leveson’s book “Safeware”.

The scope does not include System Safety Engineering except as it interfaces with the development of safety-critical software.

The bibliography particularly recommends certain sections of Prof. Nancy Leveson’s book “Safeware”.
A general purpose computer with software is the modern method of implementing a special-purpose machine. Safety-critical systems are not usually visible as computers -- only as a specialized machine.

A good example is the automotive ignition system which formerly existed as a vacuum-driven spark advance assembly. Now it is a computer with safety-critical software. The software is safety-critical because of the system which it controls -- a high-speed automobile.

If you have been frustrated when your mechanic blamed the “computer” and gave up, you realize that few practitioners understand what the computer is doing. The spark advance assembly was easy to understand and troubleshoot.

Software is often cheaper -- especially in cost of replication-- but not always. The true costs are often hidden.
IEEE-CS/ACM SEEPP task force produced -- Prioritized:
1. Public – Consistent with public interest
2. Client & Employer – Best interest
3. Product – Meets professional standards
4. Judgment – Maintain integrity & independence
5. Management – Ethical approach to product development
6. Profession – Advance the integrity & reputation
7. Colleagues – Fair & supportive
8. Self – Lifelong learning & ethical practice

Source: IEEE Computer Society

PREAMBLE
The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:
1. PUBLIC - Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.
8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

For full document see http://www.computer.org/certification/ethics.htm

A useful addition would be the ethical obligation to maximize value to the client in the software produced. See Why Software is Valuable – following.

The code of ethics is an excellent starting place which defines the objectives but is silent on the practical means to making it effective in professional practice – either academia or industrial developers.
Most safety-critical systems are not obvious computers. In addition to the ignition system, most digital displays operate under computer control. The recent move to the “glass cockpit” in modern aircraft usually implies SC software elements because the pilot must rely on the display for critical aircraft data. These displays are also common in modern automobiles but are probably less critical.

Medical instrumentation -- both diagnostic and therapeutic -- are often dependant on computer control. The Therac 25 case discusses this in detail (see Leveson. Safeware).

Air & highway traffic control is more obvious. It is easy to visualize the consequence of incorrect control instructions.

Many modern aircraft are completely dependent on computer control in order to fly. Even these systems always have backup in the case of failure, a more subtle hazard exists when the system continues to function -- but with intermittent or anomalous behavior. The publicity about the unexplained 737 crashes may be describing an example of this type of subtle failure.

The IEEE Computer magazine presents an opposing view on the SWE licensing issue in the 1/2004 issue “A Tale of Three Disciplines...”. It argues persuasively that the lack of professionalism characterizing current software development practices is largely responsible for the low quality and appalling economic waste to say nothing of the disregard for public safety.

Licensing is important because it must be present in some form to establish standards for ethical practice.
Business Survival is Software Dependant

On-line computers critical to business operations
- Mandatory recovery in seconds or minutes common
- Delayed restoration can cripple or bankrupt an organization

Internet based business growing -- Banking, stock trading, on-line commerce, etc.
- Worst-case design vital to avoid financial disasters
- Complex web services require sophisticated recovery methods
- Disasters originate from malice, overload, chance events, etc.

Insurance companies now require formal recovery planning

Explosion in lawsuits involving computers & software
- Forensic consulting business percentage doubling each year
- Failure to resolve expectation conflicts triggers litigation
NIST Report – Software Errors Impact Economy

NIST report estimates SW errors cost $60B annually
   Significant impact on US Economy

“Impact of SW errors is enormous” – NIST Director Bement

“. Every business in US now depends on SW ..”

“80 % of development costs .. correcting defects”

Some are catastrophic:
   2/98 failure of telephone service on East coast
   Cascading power failure in NE & MW involved software failure

Estimate improved testing methods save ~$22B/Y

Source: NIST Research Triangle Institute report 02-3

Not present in the report but an excellent example of the potential for disaster, was the cascading power failure in the east and mid-west. Software was involved because many of the power management mechanisms were mechanized in SW and were inadequate to the challenge. A critical mass of these mechanisms had the effect of making it worse and longer-lasting. However you view it, it is an engineering failure – design, implementation, testing, etc.
Future of Software Engineering

Functional size & complexity increasing rapidly

- Size increase ~ 10x every 5 years
- Scale matters in all engineered systems

Humphrey's analogy with transportation system's speed

"Increasingly software [i.e., computer systems] .. crucial part of the products and services in almost all industries.”

"Most computer systems .. interconnected ..”

".. more internal and external threats …”

"In .. past, .. assumed a friendly .. environment.”

Source: Humphrey, SEI/CMU 2002

An analogy helps to understanding why size and complexity is important. The importance reflects on the standard of care needed for systems of larger size and complexity. Practices and skills adequate for a system of size 10, are inadequate for size 100 or larger. Many disputes have their origin in the fact that parties to the contract have ignored this simple fact.

Watts Humphrey suggests a transportation system analogy. Consider the technology involve in transportation systems supporting three to five MPH (e.g.,foot-powered). This technology is all about shoes.

Above 10 MPH, wheels are needed unless you are an Olympic-caliber athlete. Above 100 MPH, wings and aerodynamics become the dominating technology.

Business value contained in the various forms of information systems assets in a typical enterprise will become dominant. This impacts both the value of the enterprise and the legal issues involved when losses occur for any reason.

Interconnectedness is implemented in Telecommunications in its various forms (local and long distance telephony, local computer networking, distributed networking -- national & global -- and internet interfaces) are all changing daily in this area of business. A useful example is the emerging importance of voice-over-Internet-protocol (VOIP) where conventional telephone service is routed over the internet instead of conventional long distance trunks. With this technology, critical business communications will now be dependent on the availability of of internet services which may not have any clear contractual obligations to the party being affected.
Common Computer-Based Issues in Litigation

Unacceptable performance in systems contracts
  Unclear objectives / Missed deadlines
  Unacceptable quality and reliability
  Business interruption caused by design, implementation & installation errors

Improper use of intellectual property
  Unauthorized use of designs or software
  Improper licensing
  Theft of trade secrets

Computer vandalism -- Hacking, sabotage

Retrieval of critical data at issue in litigation
The Software Engineering Debate

“A Tale of Three Disciplines …” – Poore

“Licensing Software Engineers in Canada” – Parnas

IEEE-CS Certified Software Development Professional (CSDP)

“Should Software Engineers beLicensed”, Knight & Leveson

“ACM’s Position on the Licensing of Software Engineers” – White & Simons
All manage vast complexity

Circuit engineering – Uses established components with documented design trail

Genetic engineering – Depends on toolsets & rigorous testing to manage societal risk

Software engineering – Progressed by dint of nonconformist intellect & vast expenditures rather than by disciplined science & engineering

Foundation builders have not had universal impact

SWE uses little computer power compared to circuit & genetic

General acceptance of low quality

Source: Poore, Computer 1/2004

Foundation builders named by Poore are Basili, Brooks, Boehm, Mills & Parnas.
“A Tale of Three Disciplines …” Cont.

Certification of products
  Immediate need – Public safety related
    Medical, flight management, power control, etc.

Licensing of practitioners
  SEBOK to establish education minimums
    Need consensus on curricula
    Formal enforcement of standards necessary

Accreditation – Blessing & Curse
  Too soon stifles advances
  Too late causes sub-optimal results
“Licensing Software Engineers in Canada”

Must become a self-regulating profession

- Engineering regulations responded to exploding steam boilers and collapsing bridges

Software now critical to designs impacting public safety

Computer science treats SWE as an area of research, not a profession

Use of the “engineering” title is separate from limits to academic freedom

CS chooses to argue against regulation

- Better to define appropriate criteria – e.g., SEBOK

Source: Parnas, Communications of ACM 11/2002
IEEE-CS Certified Software Development Professional (CSDP) Overview

Mod 1 – SWE & Society
Mods 2&3 – Process & Requirements
Mods 4&5 -- Design Concepts & Construction
Mods 6&7 – Testing & Maintenance
Mods 8&9 – Management & Measurement
Mod 10 – Supporting Processes; CM, QA, V&V, Reviews

Source: CSDP Preparation Course -- IEEE CS

Note that programming and coding are not considered as key elements of the profession – any more than carpentry is a critical skill for a structural engineer.
“Should Software Engineers be Licensed”

Implications of Licensing

Courts have not yet made SWE liable for malpractice

Negligence is failure to perform within the standards of the profession

SWE is not licensed, thus not subject to malpractice

Unless professional standards are very clear, courts may decide, causing harm

Conclusion: Licensing is not recommended. Other approaches need to be evaluated.

My summary of this argument -- *If there is no standard-of-care, you can’t be held accountable legally.*

This is not an ethically-defensible solution.

Source: Knight & Leveson, Communications, 11/2002
“ACM’s Position on the Licensing of Software Engineers”

Is ACM against the licensing of SW Engineers? --- YES

Primary licensing is becoming a Professional Engineer (PE)
   Current PE’s must understand such general engineering as thermodynamics, fluid mechanics, etc.
   Beyond the scope of SWE

SWE license can be interpreted as authoritative statement – Able to produce SW of consistent
   Reliability
   Dependability
   Usability

Current state of knowledge & practice is too immature to give such assurances

My View – The admission of immaturity and the implicit acceptance of this state is not an ethically-acceptable response from our professional leadership.


The is their real answer to this issue.
Software Engineering is Different

Why Software is Valuable

Significant Differences with Software Systems

Specifying Software
Why Software is Valuable

Value derives from the abstraction of productive knowledge
   SW Development is a SOCIAL learning process
Any economic value comes from impact on the useful activity it affects
   Efficient automotive ignitions
Value is increased when the knowledge is readily adaptable
   McDonalds hamburger franchises also work well in China
Franchises show how preserved abstractions can be valuable
Software engineers are ethically obligated to optimize value

Source: Baetjer
The nature of software is that it is an abstract representation of process steps that must be interpreted by another system. There are a myriad of conversions necessary to convert it into reality. Validating these conversions requires more discipline than hardware equivalent systems because it is viewed as being of less importance, i.e., it is “soft”.

Since it is abstract, visibility is never direct and is usually made more complicated by undisciplined implementation practices. The originator is often the only one that understands the design -- and then only for a limited time. This can limit the effectiveness of review and testing.

Nancy Leveson notes in “Safeware” that there are no natural bounds to the complexity which limit physically based systems. As a result, the implementation tends to become arbitrarily complex unless explicit practices inhibit it.

Since the realization of a software system does not require understanding by any but the implementer, often they are the only ones that do. As a result, errors are easily hidden and are hasty designs can be fixed without embarrassment. On the other hand, physical systems require clear and error-free design to be communicated to other parties which participate in the creation. If errors are present in a drawing or materials list, they must be corrected. Poorly fitted parts expose the designer’s mistakes and mechanisms which do not operate properly are easily noticed. This tends to encourage higher quality work because mistakes are easily seen. With software, practices naturally observed in other disciplines tend to be ignored at the working level because they can be skipped so easily.

Requirements are seldom complete - IKIWISI

“With software the challenge is to balance the unknowable nature of the requirements with the business need for a firm contractual relationship.” -- Watts Humphrey

Most engineered systems are defined by comprehensive plans and specifications prior to startup. Few software-intensive systems are.

Few natural disciplines

Inherent disciplines of hardware absent

Complexity has no inherent bounds -- Few, if any, understand the complete design

Implementation often understood only by originator -- Easy to hide mistakes, influences litigation strategy
Vital point is that software is not safe or unsafe apart from the system. That is the meaning of an “emergent property”.

It is different from reliability but often overlaps. Also safety may conflict with reliability.

• The SW may be perfectly reliable -- operating in conformance with requirements (Chem Expl) -- but be unsafe.

• Safety is never a component characteristic, reliability can be.

• Safety is usually improved by reduction in complexity -- This may only be achieved at the expense of reliability afforded by redundancy, etc.

Time is always a factor. Deterministic control of time is usually required in control-type SC SW. For instance, gain of a SW control loop is usually a direct function of the execution rate -- anything which increases or decreases the execution rate affects the gain to that extent. Many subtle and intermittent things can cause this to happen.

SW mechanization consists of millions of small state changes to achieve a larger change. If anything interferes with the sequence, an intermediate state is the result.

I may be easy to change SW but, because of its complexity and lack of visibility, it is difficult to change correctly and safely.

Clear physical laws and inherent limitations are usually absent in SW

The earlier that the safety requirements are defined, the result is better and cheaper -- design-in not add-on.

Good safety engineering is expensive -- resources must be focused on the most critical. In turn, the most critical must be partitioned from the less critical.

These methods help with early definition and isolation of the critical requirements.
Lessons-Learned

Automotive Ignition – Unsafe Design
Failed System Implementation

The case studies described are two published accounts and one personally experienced -- the automotive ignition.
Automotive Ignition Example – Unsafe Design

Engine died when accelerating into traffic
  Intermittent sensor wire
  Ignition control software failed with open circuit values

Hazard analysis missed HW-SW interaction

Incomplete system safety requirements on SW
  Interface failure protection - From Hazard analysis
    Design deterministic result for common failures -- Open, short, etc.
  Control algorithm must be protected - Derived for SW
    Detect failures and substitute “safe” values

This is a good example of the need for system-level safety requirements:
  • The ignition control shall not permit invalid readings from sensor failures to cause the ignition to fail.
  • The sensor inputs will be terminated in a manner to establish a predictable value if the line is open or shorted.
  • The SW shall test for these “open” or “Short” values and substitute a “safe” value such as the last valid input.
  • An error condition indicator shall be set when the sensor failures are detected.
Failed System Implementation

International contract to supply a set-top box to play educational software in public schools
  Missed critical deadline for school-year budget process
  Missed critical cost limits for primary customers -- schools
  Unable to play interactive software content as required
  Unable to interface with critical network standard

Unrealistic commitments by supplier management
  Ignored technical inputs from developers

Dishonest project status communications

Major losses for both parties -- Caused lawsuit
  Settled with no compensation by either party, more losses
Correcting the Problem

Framework for Dependable Designs

Example of Dependable ignition System

Safe SW begins with adequate requirements.
Examining the differences, a framework of methods and practices is presented.
Specify the complete operating envelope at needs stage.
Define recoverable system anomalies -- Include degraded normal envelope. Examples range from power recovery to protection against operator input errors.
Define system response to crossing envelope boundaries -- these establish top-level safety requirements.
Allocate top-level safety requirements to HW and SW during normal systems engineering process.
This is the most cost-effective time to establish system-safety/SW-safety requirements -- many options are available.

Framework for Dependable Designs

Imagine testifying in court -- Defend your engineering process*

Establish bounds for system in three states

  Operating -- Envelope for normal operations
  Non-Operating -- Normal operations not possible
  Exception -- Returning to normal after response to anomaly
    Normal may be degraded-normal

Mishap conditions occur during state transitions

Identifies system dependability requirements involving software

Suggests mishap mitigation or control -- HW or SW

* Source: Lawson
Example of Dependable Ignition System

Automotive Ignition -- Hazard identified
   Sensor wiring may fail from constant movement
   Ignition control failure may cause traffic emergency

Requirement - Recover safely from failed ignition sensor wiring

Allocation of requirement – “What if”
   HW - Terminate inputs for predictable open/short state values
   SW - Detect values indicating open/short, use last good value or known safe value

Requirements identification before design is best
   More options, usually less costly

A simple example that everyone can relate to.
Wiring is a common failure -- the system should respond safely.
Note that requirement is system-level -- independent of mechanization.
Allocation step maps requirement into component subsystems - HW & SW
Identifying need makes a low cost / highly effective solution possible.
Later identification might not be able to include termination for predictable values. A more complicated and less predictable solution might have to be implemented completely in the SW.
Summary

SW intensive critical systems are pervasive in all industrial countries

Maturity of SW practices largely inadequate

Government and legal systems are recognizing importance to public safety and economic security

Better SW engineering is possible now

Absent formal licensing, ethical obligation flows from individual engineer’s standard-of-care

Improve dependability of SW – Worst-case design

Establish control over the SW development and system installation processes – use engineering conservatism

Satisfy multiple stakeholders to achieve and maintain successful system operation

It is vital that SE catch up with the pervasive distribution of critical SW. Disciplined development is the key.
Bibliography -


Personal SW process is new.
Bibliography – II


Knight, Leveson, Should Software Engineers be Licensed?, Communications of the ACM, 11/2002

Lawson, Harold W., An Assessment Methodology for Safety Critical Systems, Lidingo, Sweden, Bud@damek.kth.se

Leveson, N.G., Safeware, Addison Wesley, 1995:

SafeWare recommended readings: CH 1-4, 7.2-7.3, 8.2-4, 9, (Mgmt 11, 12), Optionally Add Human Interface 5,6,17; Accident & Human Models 10.

Leveson, Nancy G., High Pressure Steam Engines and Computer Software, www.washington.edu, Keynote talk at the Int. Conf. on Software Engineering, Melbourne, Australia, 5/92
Bibliography - III


