"Safeware": Safety-Critical Computing and Health Care Information Technology

Robert L. Wears, MD, MS; Nancy G. Leveson, PhD

Abstract

Information technology (IT) is highly promoted as a mechanism for advancing safety in health care. Ironically, little attention has been paid to the issues of safety in health care IT. Computer scientists have extensively studied the problem of assured performance in safety-critical computing systems. They have developed a conceptual approach and set of techniques for use in settings where incorrect or aberrant operation (or results from correct operation that are aberrant in context) might endanger users, the public, or the environment. However, these methods are not commonly used in health care IT, which generally has been developed without specific consideration of the special factors and unique requirements for safe operations. This article provides a brief introduction for health care professionals and informaticians to what has been called "safeware," a comprehensive approach to hazard analysis, design, operation, and maintenance of both hardware and software systems. This approach considers the entire joint sociotechnical system (including its operators) over its entire lifecycle, from conception through operation and on to decommissioning. Adoption of safeware methods should enhance the trustworthiness of future health IT.

Introduction

Twenty-five years ago, Lissane Bainbridge coined the phrase "ironies of automation" to refer to the observation that introducing automation into a complex sociotechnical system to improve safety and performance often simultaneously introduced new problems into the system that degraded safety and performance.[1, 2] Despite this experience, the belief that advanced information technology (IT) is a critical mechanism by which to improve the safety of health care is strongly held by academics, public officials, and vendor, business, and civic groups.[3, 4, 5, 6, 7, 8, 9] The anticipated benefits of health care IT are presented in these discussions as a sort of manifest destiny—difficult, to be sure, but ultimately inevitable. While there have been many discussions about the challenges,[10] costs,[11] priorities,[10] and other planning issues[12] in implementing IT, there has been virtually no discussion about how to make health IT itself safe for patients, practitioners, and health care organizations. The irony of seeking safety through systems that may not be safe to begin with seems to have been lost in the enthusiasm for remaking health care via IT.

Past experience with IT has not shown it to be an unequivocal success.[13, 14] Hardware failures have propagated in unexpected ways to remote, ostensibly unrelated components on a common network[15, 16]; system upgrades have lead to missing[17] or false laboratory information[18]; programming mistakes have similarly led to incorrect guidance in decision support[19]; and computerized provider order entry, the ”Holy Grail” of safety efforts, has led to new forms of failure.[20, 21, 22, 23] These problems have led to a small but slowly growing realization that the hazards of implementing IT in a field as complex as health care have only just begun to surface. However, even this awareness has been limited, as the focus has been almost entirely on problems related to the human-computer interface[20, 22] and unintended consequences due to changes in work practices.[21] The problem that the technology itself might be inherently unsafe—that it might lead to adverse outcomes due to internal faults, interactions with users or external devices, or even when the system is operating as intended by the programmers and in the absence of human factors or work practices problems—has barely been recognized in these discussions.

Technology-related safety problems have been well recognized in the computing world, however, and a substantial body of work has developed since the 1980s concerning IT safety in safety-critical systems.[24, 25, 26, 27, 28, 29] In a wide variety of domains (e.g., military, aerospace, nuclear power, rail transport), a systematic approach to identifying, reducing, or mitigating computer-related risks is now considered standard, and the burden of proof for demonstrating that a system (or modification to a system) is safe is placed on the vendor, developer, or implementer, rather than being placed on the customer to demonstrate that it is not safe.[30]

It seems ironic that the movement to promote safety through IT seems uninformed by the field of safety-critical computing. As an example, an informal search of the *Science Citations Index* for references to three well-known texts related to computing safety[25, 27, 28] revealed no citations in medical informatics or clinically relevant journals (of 211 total). While this is by no means a definitive mapping of the relationship between the two fields, it seems reasonable to conclude that there is little evidence for their interaction. Although questions have occasionally been raised about the safety of clinical IT, they have been couched primarily in the form of concerns about whether and for what types of systems regulation should be required.[31, 32]

This lack of awareness of the field of safety-critical computing stems, in part, from the origins of health informatics, which arose at a time in which a major question was whether it was possible that IT could contribute usefully to health care. This gap has been exacerbated by a gradual change in IT usage in health care, as systems that were originally developed either as business-related, transaction-oriented systems or as research-related ”proofs of concept” have slowly and subtly become transformed into safety-critical, process-control systems.

The development, operation, and maintenance of such systems differ in important ways from common IT practices. Thus, health care faces the hazard of having safety-critical processes dependent on IT systems that are not designed, operated, and maintained using safety-critical methods. Finally, and more subtly, the changes in human and system behavior caused by the introduction of IT into health care and the unsafe conditions that may result as a consequence are difficult to envision and so often go unaddressed.

The purpose of this article is to bridge the gap between two scientific communities that could mutually inform each other’s work in a synergistic way, by:

- Introducing the field of safety-critical computing to researchers, developers, and practitioners in health care who are interested in using health IT to advance safety and quality but may have been unaware of its existence.
- Outlining some basic principles and practices of safety-critical computing.
- Guiding readers who wish to become more knowledgeable about safety-critical computing to additional resources for more detailed study and application of these methods.

Case Study

To illustrate the types of safety problems that might be intrinsic to IT (compared to arising from human-computer interface or work practice issues), we briefly review a case study (described in greater detail by Cook and O'Connor[33]).

On a Friday night shift in a large, tertiary care hospital, a nurse reported to the pharmacy that the medications just delivered to the floor in the unit dose cart for a particular patient had never been ordered for that patient. While they did match the recently printed Medication Administration Record (MAR), comparison to the previous day's MAR showed substantial changes, and there was no record in the chart of any relevant orders. The pharmacy's computer records for the patient in question matched the recent MAR, but before the discrepancy could be understood, more discrepancies from other nursing units, in all areas of the hospital, began to be reported; all concerned drugs that matched the current MAR in the computer but were wrong for the patient.

As the magnitude of the problem mounted, the pharmacy technician called a senior pharmacist who realized that a serious, hospitalwide crisis was upon them. The computer system was somehow producing an inaccurate fill list, such that neither the MAR nor the unit dose carts already on the wards could be trusted. Early Saturday morning, a plan was devised to send all the unit dose carts back to the pharmacy and to manually recreate the deliveries needed from the previous day's printed MARs and the handwritten orders in the charts. These manual procedures enabled the hospital to continue functioning through the next 24 hours, albeit at great effort, until the system could be repaired and then "brought up to date" so that its internal representation of the "patient-medication state" of the hospital matched the external world. As far as is known, no patient suffered serious harm from this event.

The ultimate explanation for the event involved multiple factors. The pharmacy software was from a major vendor but had been customized with a special dose-checking routine in the aftermath of a severe accident. It had not been upgraded regularly due to the need to rewrite and retest these special procedures after each upgrade. Extensive backup procedures were in place and operational. On the day of the event, the software had detected a database fault, which the hospital's IT department and the vendor attempted to fix. The fix did not work satisfactorily, so part of the database was reloaded from the most recent backup tape; after this, the system appeared to function correctly.

Due to a complex technical problem in the database management software unrelated to the previous fault, the reload was incomplete in ways that were not apparent to the operators, leaving the MAR database internally corrupted. Fundamentally, the system had performed as designed, but the design had not anticipated the set of circumstances that led to internal database corruption and did not have the capability to detect or respond to such damage.

Ironically, one critical factor in the successful recovery was that the entire system was not automated. Correction and recovery would have been much more difficult if not only the MAR system, but also the order entry and the dispensing functions, had been integrated into the same flawed system.

This case illustrates a number of important safeware principles: (1) normal operations are no assurance of correct operation; (2) testing has limited value in establishing system safety; and (3) system maintenance can be a major hazard in complex systems.

Safeware Principles: A Brief Outline of the Safeware Approach

Safety in software-intensive systems has been a concern in other industries for decades, and much has been learned about how to introduce IT into a safety-critical system. In this section, we outline some basic principles and the approach widely used in software system safety. Only a brief introduction is possible here; for more information see Leveson.[25]

Guiding Principles

- The first basic principle is that safety is a system problem, not a software or IT problem. Computer behavior that may be perfectly safe in one system context may be unsafe in a different environment. Computers are not an inherently dangerous technology, in the sense that, say, petrochemical refining is inherently dangerous. Computers do not present hazards directly but rather only become unsafe when used in an environment in which mishaps and unacceptable losses can occur. Therefore, building and ensuring safety starts at the system level, not the component or software level.
- A second principle is that safety and reliability are not only different properties, they are sometimes conflicting. Reliable software (i.e., software whose performance is invariant) is not necessarily safe, and safe software does not have to be reliable. In some instances, increasing reliability can actually decrease safety (e.g., the computer continues to do something even though that behavior is unsafe in the current environment, and vice versa, the safest behavior under certain conditions may be to stop operating and switch to some fail-safe mode). In addition, “failing” (i.e., discontinuing operation) is not the most important safety issue with software. Most accidents are caused not by the computer stopping, but by it operating but doing something unsafe. It is relatively easy to protect the system against total failure, but it is much more difficult to protect it against unsafe software operation.

The field of system safety (and software system safety) has its roots in the intercontinental ballistic missile (ICBM) systems of the 1950s, when very complex, largely autonomous, software-intensive systems were built, in which accidents would have catastrophic consequences that could not be blamed simply on human operators. System safety is a subdiscipline of system engineering and encompasses many of the same principles:

- Safety must be built into a system from the beginning; it cannot be added to a completed design or tested into a system.
- Accident and loss prevention require a top-down approach that deals with systems as a whole and not just components of the system.
- Accidents are not caused by component failures alone. In fact, in software-intensive systems, accidents are much more likely to result from dysfunctional and unsafe interactions among normally operating (not failed) components.
- Accidents can be prevented using hazard analysis and design for safety to eliminate or control hazards.
- In software-intensive, complex systems, qualitative rather than quantitative approaches need to be emphasized as quantitative procedures must necessarily omit important but unmeasurable factors and therefore may be misleading.

System safety starts from hazards and emphasizes hazard analysis and control as a continuous, iterative process applied throughout system development and use. Once hazards have been identified, they are handled by either elimination from the system design if possible, or if not, by preventing or minimizing their occurrence, controlling them if they occur, and minimizing damage (in that order).

As an example, consider a computer-controlled analgesia or insulin pump (both of which, historically, have been involved in serious patient injury and death). The most critical hazard is administration of unsafe levels of the medication. Eliminating the hazard might be possible by substituting a less dangerous drug. If that’s not possible, then steps must be taken to prevent an overdose, to detect and counteract it if it occurs, and to initiate emergency treatment to minimize damage from an overdose.

To provide this protection, a formal system safety process can be used. Surprisingly, the use of such a process is not expensive, and it may be much less costly than applying expensive and ineffective testing and assurance programs. The rest of this section briefly describes the system safety and software system safety process.

Safety Over the System Life Cycle

The guiding principle for this approach is that safety must be designed into the system and software from the beginning. Attempting to add it to a completed design is not only extremely expensive, but it also is not very effective. Obviously, this is a significant issue for clinical systems that have been created by layering them over pre-existing business or research systems in which safety was not an important concern. To be most effective, system safety needs to be considered during program/project planning, concept development, system design, system implementation, configuration control, and operations. The tasks associated with these life cycle stages are technically complex and cannot be described briefly; readers should consult more detailed works or seek expert assistance in these areas. The following provides a guide to these activities:

During program/project planning. Develop safety policies and procedures and specify a system safety plan. This plan includes how software safety will be handled. Construct a system safety management structure—including well-defined authority, responsibility, and accountability for safety—and define appropriate communication channels for safety-related information. Ideally, keep the safety management system and team separate from the development system and team. This structure must include responsibility, accountability, authority, and communication channels for the IT developers as well as the system developers. Finally, create a safety information system, including a hazard tracking system.

During concept development. Identify and prioritize hazards (typically using severity). As architectural designs are considered and selected, elimination and control of hazards will be a major decision factor. Once the architecture is defined, specify safety-related system requirements and constraints for the development and operation of the system.

During system design. Hazard analysis is applied to the design alternatives to:

- Determine if and how the system can get into the hazardous states.
- Eliminate the hazards from the system design, if possible.
- Control the hazards in system design if they cannot be eliminated.
- Identify and resolve conflicts among design goals using safety as one of the decision criteria.

After the system safety analysis and design are complete. Trace unresolved hazards to the system components, including hardware, software, and humans. Generate safety requirements and constraints for each of the components from the system safety requirements and constraints.

During system (and component) implementation. Design safety into the components (using the safety requirements and constraints provided as a guide) and then verify the safety of the constructed system. It is important to note that testing for safety, particularly for software systems, is not practical. Only a very small part of the entire software state can be tested. Accidents usually occur when factors have been forgotten or not accounted for in the software or system design, and those same factors will almost surely be omitted in testing as well.

There are some aspects of safety that should be tested (e.g., special processes or procedures for handling specific hazards), but little confidence can be placed on the results. One cannot test safety into a system. Designing safety into software may involve such software engineering practices as defensive programming, assertions and run-time checks, separation of critical functions, elimination of unnecessary functions, exception handling, and others.

At implementation. Use the documentation developed as a by product of safety management during development to produce a formal "safety case" argument for the safety of the system. Such formal analyses of system safety are now required for safety-critical systems in industry in the United States and European Union[34] and should be continuously maintained and updated as experience is gained in operations (including near misses and accidents), thus becoming a continuing argument for system safety.[35]

During operations and maintenance. Evaluate all proposed changes for safety using the same hazard analyses and assumptions (that should have been recorded) used during development. However, changes are not always planned, so periodic audits and performance monitoring are required to verify that the assumptions underlying the safety analysis used in the development process still hold.

Finally, incident and accident analysis is clearly necessary, which implies that there is a way to detect and communicate when safety-related incidents occur. Feedback must be established to ensure that human behavior is not changing over time in a way that could violate the system safety assumptions used during development and to check for other types of changes that could lead to mishaps.

Further Considerations

Sometimes an assumption is made that if software has been executing safely in one environment, it will be safe in another environment. This assumption is false. In fact, most software-related accidents have involved reused software. Safety is a system property, it is not a component property; software that executes perfectly safely in one environment can be hazardous in another.

Not only must the environment be analyzed for safety during initial development (as discussed above in terms of identifying and controlling hazards), but also the potentially hazardous behavior of any reused software must be carefully analyzed and evaluated and controlled. In many cases, it will be cheaper and safer simply to reimplement the software for the new system.

Commercial off-the-shelf (COTS) software presents a particular dilemma. Some companies producing medical software and IT often assume that they can simply provide a version that everyone can use. This is not realistic. IT must be tailored for the larger system in which it will operate (which always has unique features), and the safety of the system (in the proposed environment) has to be carefully analyzed. In addition to their own vulnerabilities, COTS systems bring unknown vulnerabilities into the operating system employed (typically in many different versions) and the hardware platform (similarly, in many different versions).

A Guide to Further Learning

Safety in computer systems is a large and active field. In this section, we suggest several sources that should be useful for those interested in safer IT or a more detailed exposition of safeware principles.

- *Computer-Related Risks*[27] is a classic text on accidents and near misses related to faults in IT systems. *The Risks Digest*[36] can be viewed as an online continuation of this work. It is moderated, regularly updated, searchable, and well regarded in the computer science world.
- *Safeware: System Safety and Computers*[25] has become the classic text on IT safety and gives a detailed exposition of principles, specific applications, and analyses of a set of well-known IT accidents.
- *Safety-Critical Computer Systems*[28] is a similar text, not quite as comprehensive and with a bit more focus on embedded systems.

- *Trust in Technology: A Socio-Technical Perspective*[29] provides an overview of the nature of safe, trusted IT and how it might (or might not) be implemented and operated.

Conclusion

The traditional approach to producing software is to determine the requirements, implement them, and then try to assure that there are no errors in either. The problems with this approach from a safety standpoint are that correct implementation of the requirements does not guarantee safety, and it is impossible to ensure that software is "perfect." In fact, perfect (error-free) software does not exist. The alternative proposed in this article is to begin by envisioning the states the system should never get into and then work backward to design, implement, maintain, and operate the system such that either those states cannot be reached, or if they are reached, they are detected and handled safely before losses occur.

One serious problem related to the safety of health IT is the as yet unanswered question of who is responsible to see that systems are designed, implemented, operated, and maintained with safety as a central feature. Currently, a good deal of health IT is not subject to regulatory oversight; purchasers have little leverage in negotiations with vendors on safety issues; and both purchasers and vendors may have limited understanding of the hazards of such systems and effective means of managing them. This is an important social (ethical) regulatory issue that will need to be addressed constructively in order to ensure that these principles are actually applied in health IT. Health care would do well to follow the lead of other hazardous industries and develop ways to ensure that the burden of proof for demonstrating that a system is safe is placed on the vendor, developer, or implementer, rather than being placed on the customer to demonstrate that it is not safe.[30]

Key Messages

The key messages of this article are:

- Safety is a system problem, not just an IT problem.
- While IT has the potential to greatly improve quality and safety in medicine, that result is not guaranteed; the way in which IT is designed, implemented, maintained, and operated will determine what kind of result ensues.
- To ensure that safety is improved and not inadvertently compromised requires a systems approach and analysis of the overall sociotechnical system in which the IT will be embedded.
- Much has been learned about how to design, implement, maintain, and operate safety-critical IT in other settings; that knowledge could usefully inform attempts to introduce IT into health care to advance safety goals.

Author Affiliations

University of Florida, Imperial College London (Dr. Wears); Massachusetts Institute of Technology (Dr. Leveson).

Address correspondence to: Robert L. Wears, MD, MS, Department of Emergency Medicine, Box C-506, University of Florida Health Science Center, 655 W. 8th Street, Jacksonville, FL 32209; e-mail: wears@ufl.edu.

References

1. Bainbridge L. Ironies of automation; 1983. Available at: www.bainbrdg.demon.co.uk/Papers/Ironies.html. Accessed February 29, 2008.
2. Bainbridge L. Ironies of automation: Increasing levels of automation can increase, rather than decrease, the problems of supporting the human operator. In: Rasmussen J, Duncan K, Leplat J, eds. New technology and human error. Chichester, UK: Wiley; 1987. p. 276-283.
3. Bates DW, Cohen M, Leape LL, et al. Reducing the frequency of errors in medicine using information technology. J Am Med Inform Assoc 2001; 8: 299-308.
4. Bobb A, Gleason K, Husch M, et al. The epidemiology of prescribing errors: The potential impact of computerized prescriber order entry. Arch Intern Med 2004; 164: 785-792.
5. Executive Order: Promoting quality and efficient health care in Federal Government administered or sponsored programs. Available at: www.whitehouse.gov/news/releases/2006/08/print/20060822-2.html. Accessed February 29, 2008.
6. Rosenfeld S, Bernasek C, Mendelson D. Medicare's next voyage: Encouraging physicians to adopt health information technology. Health Aff (Millwood) 2005; 24: 1138-1146.
7. Taylor R, Bower A, Girosi F, et al. Promoting health information technology: Is there a case for more aggressive government action? Health Aff (Millwood) 2005; 24: 1234-1245.
8. Milstein A, Galvin RS,Delbanco SF, et al. Improving health care safety: The Leapfrog initiative. Nov-Dec 2000; Effective Clinical Practice. Available at: www.acponline.org/journals/ecp/novdec00/milstein.htm Accessed May 27, 2008.
9. Leapfrog Group. The Leapfrog Group fact sheet. Available at: www.leapfroggroup.org/media/file/leapfrog_factsheet.pdf. Accessed February 29, 2008.
10. Jha AK, Poon EG, Bates DW, et al. Defining the priorities and challenges for the adoption of information technology in healthcare: Opinions from an expert panel. AMIA Annual Symposium proceedings [electronic resource] /AMIA Symposium 2003: 881. Available at: www.amia.org/pubs/proceedings/symposia/2003/265.pdf. Accesed February 29, 2008.
11. Kaushal R, Blumenthal D, Poon EG, et al. The costs of a national health information network. Ann Intern Med 2005; 143: 165-173.
12. Ash JS, Stavri PZ, Kuperman GJ. A consensus statement on considerations for a successful CPOE implementation. J Am Med Inform Assoc 2003; 10: 229-234.
13. Johnson CW. Why did that happen? Exploring the proliferation of barely usable software in healthcare systems. Qual Saf Health Care 2006; 15(suppl 1): i76-i81.
14. Rochlin GI. Trapped in the net: The unanticipated consequences of computerization. Princeton, NJ: Princeton University Press; 1997.
15. Perry SJ, Wears RL, Cook RI. The role of automation in complex system failures. J Patient Safety 2005; 1: 56-61.
16. Wears RL, Cook RI, Perry SJ. Automation, interaction, complexity, and failure: A case study. Reliability Engineering and System Safety 2006; 91: 1494-1501.
17. Wears RL. More on computer glitches and laboratory result reporting. Available at: catless.ncl.ac.uk/Risks/23.64.html#subj4. Accessed February 29, 2008.

18. Burstein D. Canadian medical tests give reversed results. Available at: catless.ncl.ac.uk/Risks/23.19.html#subj2. Accessed February 29, 2008.

19. Northern General Hospital NHS Trust. Report of the inquiry committee into the computer software error in Downs syndrome screening. Available at: www.sheffield.nhs.uk/resources/Downs-report.pdf. Accessed February 29, 2008.

20. Horsky J, Kuperman GJ, Patel VL. Comprehensive analysis of a medication dosing error related to CPOE. J Am Med Inform Assoc 2005; 12: 377-382.

21. Han YY, Carcillo JA, Venkataraman ST, et al. Unexpected increased mortality after implementation of a commercially sold computerized physician order entry system. Pediatrics 2005; 116: 1506-1512.

22. Koppel R, Metlay JP, Cohen A, et al. Role of computerized physician order entry systems in facilitating medication errors. JAMA 2005; 293: 1197-1203.

23. Baldwin FD. Physician resistance arrests CPR system. In: Healthcare informatics; 2003: 34-36.

24. Leveson NG. Software safety: Why, what, and how. ACM Computing Surveys 1986; 18: 125-163.

25. Leveson NG. Safeware: System safety and computers. Boston: Addison-Wesley; 1995.

26. Leveson NG. A systems-theoretic approach to safety in software-intensive systems. IEEE Transactions on Dependable and Secure Computing 2004; 1: 66.

27. Neumann PG. Computer-related risks. New York, NY: ACM Press; 1995.

28. Storey N. Safety-critical computer systems. Harlow, UK: Pearson Education Limited; 1996.

29. Clarke K, Hardstone G, Rouncefield M, et al. eds. Trust in technology: A socio-technical perspective. Dordrecht, Netherlands: Springer; 2006.

30. Jackson D, Thomas M, Millett LI, eds. Software for dependable systems: Sufficient evidence? Washington, DC: National Academies Press; 2007.

31. Coiera EW, Westbrook JI. Should clinical software be regulated? Med J Aust 2006; 184: 600-601.

32. Miller RA, Gardner RM. Recommendations for responsible monitoring and regulation of clinical software systems. J Am Med Inform Assoc 1997; 4: 442-457.

33. Cook RI, O'Connor MF. Thinking about accidents and systems. In: Manasse HR, Thompson KK, eds. Medication safety: A guide for healthcare facilities. Bethesda, MD: American Society of Health Systems Pharmacists; 2005: p. 73-88.

34. Reason J. Managing the risks of organizational accidents. Aldershot, UK: Ashgate Publishing Co; 1997.

35. Kelly TP, McDermid JP. A systematic approach to safety case maintenance. Reliab Eng Syst Safe 2001; 71: 271-284.

36. Neumann PG. The risks digest. Available at: catless.ncl.ac.uk/Risks/. Accessed February 29, 2008.