SANDIA REPORT

SAND98–2737
Unlimited Release
Printed February 1999

Managing Errors to Reduce Accidents in High Consequence Networked Information Systems

John H. Ganter

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

SAND98-2737
Unlimited Release
Printed February 1999

Managing Errors to Reduce Accidents in High Consequence Networked Information Systems[a]

John H. Ganter
Decision Support Systems Software Engineering
Sandia National Laboratories
P. O. Box 5800
Albuquerque, New Mexico 87185
jganter@sandia.gov

ABSTRACT

Computers have always helped to amplify and propagate errors made by people. The emergence of Networked Information Systems (NISs), which allow people and systems to quickly interact worldwide, has made understanding and minimizing human error more critical. This paper applies concepts from system safety to analyze how hazards (from hackers to power disruptions) penetrate NIS defenses (e.g., firewalls and operating systems) to cause accidents. Such events usually result from both active, easily identified failures and more subtle latent conditions that have resided in the system for long periods. Both active failures and latent conditions result from human errors. We classify these into several types (slips, lapses, mistakes, etc.) and provide NIS examples of how they occur. Next we examine error minimization throughout the NIS lifecycle, from design through operation to reengineering. At each stage, steps can be taken to minimize the occurrence and effects of human errors. These include defensive design philosophies, architectural patterns to guide developers, and collaborative design that incorporates operational experiences and surprises into design efforts. We conclude by looking at three aspects of NISs that will cause continuing challenges in error and accident management: immaturity of the industry, limited risk perception, and resource tradeoffs.

[a] This paper is based on a presentation at the Workshop on Information Assurance and Trustworthy Networks, held by the Cross Industry Working Team (XIWT) and Bellcore in Washington, D.C., 17-18 November 1998.

Acknowledgements

Thanks to the following people for their comments on drafts of this paper: Bryon Cloer, William K. Storage, Barry M. Leiner, Charles C. Howell, Patty G. Guyer-Stevens, Ann L. Hodges, and Craig Dean. The opinions expressed herein are those of the author and do not necessarily reflect those of any other individual or organization.

Table of Contents

This page intentionally left blank.

Introduction

Networked Information Systems (NISs) are heterogeneous computing systems consisting of multiple computing nodes linked by networks. NISs have two primary roles: (1) for communication between individuals and organizations, and (2) as monitoring and control systems for other systems and infrastructures. In these roles, NISs are increasingly critical to the operation and integrity of domains such as finance, transportation, energy, and healthcare. NISs are becoming as large, complex, dynamic, and important as older industrial, transportation, weapons, nuclear, and space exploration systems – and they are increasingly part of such systems. As a result, the failure or corruption of NISs can have large consequences as measured in money, customer and public confidence, and lives.

NISs must be *trustworthy,* an aggregation of attributes such as correct, reliable, safe, secure, and survivable (Schneider 1999, p. 14). Trustworthiness is a challenging design goal because of complex interrelationships among these attributes. For example, increasing reliability through redundancy can reduce security, or it can have other effects that are difficult to anticipate. Most effort to increase trustworthiness has been aimed at technical elements. For example, formal methods help to ensure that designs are complete and valid, testing helps to confirm correct construction, encryption provides security during operation, and fault-tolerance increases survivability under extreme operating conditions.

Trustworthiness also depends on human and organizational factors that are ubiquitous but that have received less systematic investigation. These factors are described by terms such as accident analysis, human error, lessons learned, safety practices, and operating procedures. Traditionally, these issues have been addressed in simple and symptomatic ways. For example, an error is discovered during operation, and then a rule is developed that attempts to prevent recurrence.

System safety is a more holistic perspective that views systems as having life spans from design through development, operation, and reengineering. Safety is considered an emergent property of the whole system over time, rather than any specific design feature, operating procedure, or human error that is successfully resolved. System safety emphasizes two analytical techniques: (1) ***Hazard analysis*** is first used to discover threats to systems that may not have been experienced yet. (2) ***Risk analysis*** is used to discover consequences of failure that may not be obvious to managers, designers, and operators. Both of these techniques can help avoid actual failures and accidents.

The purpose of this report is to document opportunities to transfer existing system safety knowledge into the domain of NISs. NIS design, development, and operation may be able to benefit from these lessons learned. This has the potential to reduce costs, save time, and increase user confidence in the future development and operation of NISs.

In the next section we will look at concepts for describing failures and accidents in systems. We will then consider categories of human error that weaken or penetrate the defenses of systems to cause or allow these events. A key idea is the accident trajectory that penetrates several layers of defenses because of relatively small actions over time. For instance, an error that occurred during past maintenance may combine with an error occurring during operation to allow an accident. Maintenance is a paradoxical defense because it is necessary in all systems, yet it can also cause or exacerbate problems if it includes errors that are not detected. We will then take a broader look at errors, and their prevention, throughout the system lifecycle. Design philosophies and architectural patterns have good potential for making system developers more aware of system challenges and successful solutions. We conclude by examining the continuous challenges to safety, in particular risk perception and investment decisions that do not prioritize safety. While safety is a laudable goal in NISs, the up-front costs are a significant deterrent to those with short time horizons.

Concepts for Describing Failures and Accidents in Systems

Systems operate by virtue of timing, proximity, and coordination of components. A ***failure*** is the inability of a component or a whole system to achieve its intended function. A failure may be a consequence of the failure(s) of subcomponents. For example, an operating system kernel may crash as a result of a network card failure. The failure may in turn produce consequences in other components or the overall system that depends on it. A failure may be the result of a human error. To continue the example, an NIS may fail as the result of the kernel crash on one of its nodes.

A failure that results in loss of system output and/or damage is an ***accident*** (Perrow 1984, p. 64). In our example, the failure of the NIS might result in an accident when the gas turbine engine that the NIS is controlling cannot operate without it. Undesirable situations do not always result in losses. An ***incident*** is "any event that could have had bad consequences but did not" (Reason 1997, p. 118). Typically, an incident occurs when hazards get within one defensive layer of causing damage or loss. Incidents can be valuable because they foreshadow accidents and open the possibility of prevention.

Time is not always linear throughout systems. Failures may be so brief that they have small effects. Or small failures may wait silently to produce larger failures, as in the case of a worn bolt that eventually fails.

In many systems, failures and even accidents are routine and tolerated because they have low consequences.[1] In a small email system, for example, it may be less expensive to accept and correct occasional accidents (perhaps with insurance) than to try to prevent them. People may perceive significant opportunity costs of *not* using systems in risky ways, e.g. waiting for a system to be made secure may result in loss of revenue (Schneider 1999, p. 188). But in ***high consequence*** systems, accidents must be minimized because of their very high costs in lives, national security, or money. With their expanding reach and impact, many NISs are beginning to move into zones of high consequences.

Many failures are the result of physical laws: things wear out and break. For example, a resistor in the network card example may have lost electrical conductivity from many cycles of heating and cooling. Other failures result from the actions, or inactions, of designers, builders, operators,

[1] Software is particularly non-linear, however. Small failures can become large failures under slightly different conditions or environments of operation.

or maintainers involved with systems. This nonmalicious human error has been blamed for 80 percent or more of industrial system failures (Reason 1997, p. 42) and is the focus of this paper.

When systems fail, we look for whatever is closest: a defective part, an unexpected interaction, or a person. This "proximity bias" produces quick results, but it often addresses only symptoms. As a result, the problem may continue or recur at some later date. In the last twenty years, system designers and critics have begun to look at system accidents more holistically or even to see them as an inherent part of the life spans of very complex systems (Perrow 1984).

The systems safety community has developed terms that help to take a longer view of accidents. ***Active failures***, those resulting from front-line individuals (or components), are most visible and receive most attention and blame.

To redirect our example towards people, consider an alternative to the failure of the resistor. A system operator enters an invalid value in an application dialog box. This active failure causes the failure of the operating system that causes the failure of the NIS that causes the failure of the turbine that causes an accident, resulting, for instance, in the loss of ship propulsion.[2] At first glance, the operator, like the resistor, is to blame.

More penetrating analysis often shows, however, that active failures are actually consequences of ***latent conditions*** that have existed for long periods (Reason 1997, p. 10). Latent conditions can include unrecognized component weaknesses and silent failures (including loss of warning systems), design and implementation errors, maintenance errors, mistaken policy, poor training, and excessive production pressures.

So blaming the operator may do little to prevent the next accident. Operators will always enter invalid values; it is the job of designers to limit the opportunities to make these errors and then to limit their effects when they inevitably occur. Blaming the operator for the failure of the NIS is unproductive also because it does nothing to prevent the next failure. In our example, it is more productive to examine the latent conditions in the system. Latent conditions include applications that accept invalid values and thus permit divide-by-zero and other classical computer faults. Other latent conditions are operating systems that allow one application to cause others, or the OS itself, to fail.

Both of these latent conditions come from human errors, but they are not close to the active failure. Instead, they are located "upstream" in the system design and development process and in management practices and decisions. As such, they are "parent errors" that will continue to combine with local conditions to produce "child failures" in many forms (Reason 1997, p. 120). Thus there is high value in focusing on reducing or eliminating the parent errors instead of active failures after they occur.

In the next section, we will examine types of human errors that produce both latent conditions and active failures.

[2] This is the apparent sequence in an accident involving the USS Yorktown, an AEGIS guided-missile cruiser; see Slabodkin 1998.

Some Terms for Describing Human Effects in Systems

People conceive, design, assemble, test, operate, and maintain systems through their individual and group actions. Most actions are positive; they are beneficial to the effectiveness and efficiency of the system. Some actions are negative; they have a detrimental effect by causing failures and accidents. Other actions are uncertain; it is not clear what effect they will have. In this section, we will consider the fragile relationship between intent (what a person wants to do), action (what a person actually does), and the system (whether the action was appropriate at the time). When this relationship breaks down, failures and accidents are the result.

A human ***error*** is the failure of intentions "to achieve their desired ends – without the intervention of some unforeseeable event" such as being struck by lightning. (Definitions in this section are from Reason 1997, p. 71.) Errors can be further classified by the deviation between intent and action. A ***slip*** occurs when the plan or intent is adequate, but the action fails. The person wants to do the right thing but does not succeed. For example, an operator at a keyboard intends to enter 9, but inadvertently pushes 0 because it is a neighboring key (see Table 1). A **lapse** is a failure of memory. The person intended to do something, but forgot. For example, the operator intended to start or stop a process but was distracted by something else and forgot to take the action that was intended.

Table 1. Types of human errors and examples from the lifecycle of a hypothetical NIS.

Type	Subtype	Designer/developer	Operator	Maintainer
Error	Slip	Keyboard error	Keyboard or field error	Cabling error, wrong field or value
	Lapse	Forgets to make a change, forgets code context, bounds checking	Forgets to make a change or start/stop a process, version control	Forgets to restore state, forgets step in procedure, version control
Mistake	Rule-based	Follows process, methodology	Follows training, SOPs	Follows training, SOPs
	• Bad rule	Unaware of change in system design	Uses obsolete SOP	Unaware of change in system configuration
	• Good rule, wrong situation	Improves code, does not perceive impact on other code	Gives user wrong permissions	Shuts down whole system for minor reason
	• Violation: fail to apply good rule	Does not follow validation procedure	Reveals sensitive information	Sets system to vulnerable state
	• Erroneous	Unaware of good practice	Unaware of new procedure	Unaware of new procedure
	• Deliberate	Skips steps to save time	Password misuse	Partial repair to save effort
	Knowledge-based	Algorithm has flaw; does not understand limits of a validation procedure	Cannot explain novel symptoms	Cannot troubleshoot novel problem

On the other hand, a ***mistake*** occurs when actions succeed, but the plan was inadequate or wrong. The person did the wrong thing well. There are two subcategories of mistakes depending on whether rules or problem solving is involved.

Rule-based mistakes center on the appropriateness of rules to situations.[3] A bad rule is a rule that is wrong, obsolete, or inappropriate. A good rule, but used in the wrong situation, is another way of making a mistake. A ***violation*** occurs when a good rule is not followed. Violations can be either erroneous (did not know the rule was in effect) or deliberate (knew about the rule but chose to ignore it).

Knowledge-based mistakes occur when a person cannot solve a problem or dilemma. Often, the stage is set when the situation is novel and there simply is no rule available. As Reason (1997, p. 74) puts it: "…the requisite variety of the procedures necessary to govern safe behavior will always be less than the possible variety of unsafe situations." At this point, people must rely on their expertise and that of others. They may need to use anomaly recognition, analogy, and mental simulation to determine what is actually occurring and to synthesize a novel response. (Examples of such real-time expertise are discussed in Klein 1998.) Often there is little time and significant performance pressure, which causes slips and lapses in this unrehearsed sequence of problem solving steps.

Errors and mistakes are the seeds of system failures and accidents. In the next section, we will consider how systems resist these effects and show that accidents almost always result from "causal contributions from many different people distributed widely both throughout the system and over time" (Reason 1997, p. 8).

System Defenses and Accident Trajectories

The problems that errors and failures can cause are seldom far from the minds of system designers. In the case of NISs, designers are cognizant of a variety of hazards that can cause damage to the far-flung, yet interdependent, components (Schneider 1999, p. 15). ***External hazards*** include casual computer vandals, fully malicious attackers, misbehaving outside applications, the physical environment (earthquakes, hurricanes), and infrastructures on which the NIS depends (power and telecommunications services). ***Internal hazards*** are failures resulting from various errors, mistakes, and violations in the design, development, operation, and maintenance phases of the NIS.

In response, designers place ***defenses*** between the system components and perceived hazards, both external and internal (Figure 1). (Figures are located at the end of the report.) In an NIS, the first line of defense is often a firewall, which limits the extent of communications between one network (e.g., a company LAN or intranet) and the hazardous outside world (the Internet). Firewalls are often one-way valves or diodes that only allow network protocols to be outbound. By limiting access, the firewall protects against many actions regardless of intent (errors, mistakes, and especially violations). Other defenses are raised against internal hazards. For example, detailed logging may be used to track changes made by authorized users so that errors

[3] There are several combinations of rules and outcomes that are not considered here. For instance, rules may be violated successfully, which can lead to overconfidence and underestimation of hazards (see Reason 1997, p. 75).

and mistakes can be detected (or at least their effects determined if they are discovered). Some defenses act against both internal and external threats; for example, operating system accounts and file system access control.

Defenses are never perfect; they contain holes. Some holes are known. They may be intentional tradeoffs between system attributes. For example, "trapdoors" are often installed in systems to facilitate maintenance at the expense of security. Other holes are only suspected, or they are unknown until discovered. Many holes are ephemeral; they open and close with the passage of time and changing conditions. For example, in one study it was found that a workstation became insecure after six weeks because of changing configuration (Schneider 1999, p. 44).

Since no single defense can be perfect, they are arrayed in multiple layers: ***defense in depth***. The reasonable assumption is that a hazard that penetrates one kind of defense will likely be stopped by another kind of defense.

Accidents occur when hazards penetrate multiple holes that are in unfortunate alignment. This ***accident trajectory*** (Reason 1997, p. 11) occurs when some holes contributed by latent conditions, and other holes contributed by active failures, coincide with a hazard (Figure 2). An investigation of such a accident might proceed along a backwards pathway. For instance, the active failure was an operator error; the operator forgot to set a software switch in the firewall (a lapse). The first latent condition was a maintenance mistake: applying the wrong patch to the firewall software (good rule, wrong situation). The parent of this condition was overworked staff. The parent of this condition was in turn an organizational decision about allocating resources. And, finally, the hazard was provided by a passing hacker.

Most NIS accident reports focus on the beginning and the end of the trajectory, that is, the operator lapse and the attacker. Ironically, these are the least productive areas for accident understanding and prevention. The question is "not why an error occurred, but how it failed to be corrected" (Reason 1997, p. 25) or otherwise caught by defenses. In this case, the organization would be wise to look for other latent conditions that have resulted from the same and similar organizational decisions. These holes will be available for future accident trajectories unless they are identified and closed.

Paradoxical Defenses: Defenses that Have the Potential To Be Hazardous

In addition to having holes, some defenses "designed to protect against one type of hazard can render their users prey to other kinds of danger, usually not foreseen by those who created them, or even appreciated by those who use them" (Reason 1997, p. 41). These paradoxical defenses are inherent in technological systems, from Reason's classical example of English knights rendered immobile by their heavy armor to a safety device that breaks off and plugs coolant flow in a nuclear reactor (Perrow 1984, p. 53). We will consider two paradoxical defenses of particular relevance to NISs: automation and maintenance.

Designers often try to reduce or eliminate the roles of humans in systems through automation. Automation has many benefits. Cellular telephones, network switches, efficient heating and cooling, medical devices, and many other technologies would be impractical without automation. But in systems that are not single-purpose, automation can introduce subtle hazards inside system boundaries and defenses.

In NISs, automated defenses (such as firewall alarms, "smart" filtering, and attack profile detection) can reduce accidents because people are too slow to respond or too easily bored to be continuously vigilant. Industrial experience, however, shows several common weaknesses of similar defenses.

Layered, automated defenses may hide latent conditions by eliminating their symptoms. For example, a defective network device may be sending bad packets, but a "smart" router automatically dumps invalid packets. Later, the router changes state, the dumping stops, and the packets cause a higher level failure in an application.

Because automation encapsulates the thinking of a designer into a machine, there are opportunities for conflicts with the other thinkers – human operators and maintainers – in the fielded system.

A common problem is misunderstanding the current operating mode. The operators think the automation is off, on, or doing one thing, but they are wrong. For example, misunderstandings of autopilots have caused a number of aviation accidents. Another problem is false alarm rates that cause protections to be disabled or ignored by operators. This latent condition sets the stage for an active failure to cause an accident – while the automation, correct for once, is ignored. The real latent condition, we will suggest below, is a design process that does not fit the automation into the operational environment.

Maintenance is essential to the long-term health of all systems. Maintenance is also implicated in many industrial accidents. In one study, for example, 55 to 65 percent of performance problems in the nuclear power industry were related to maintenance (Reason 1997, p. 92). Systems are relatively easy to take apart, but reassembly is difficult. Correctness can fall victim to slips, lapses, mistakes, and violations. Maintainers omit both actions and components and restore the system to wrong states after maintenance. Holes opened in defenses may be left open, for example, when a maintainer leaves an NIS component in debugging mode. Defenses may be removed entirely if a maintainer leaves out a command line setting or leaves a service or daemon in the wrong state. Again, we suggest that it is more productive to look upstream at high-level errors that result in latent conditions. Such high level errors include weak design, weak installation, time pressure, resource allocations, poor teamwork, and lack of training.

In the next section, we will look at the system lifecycle in more detail to see where attention to reducing errors and failures, and the lessons of accidents, can be most profitably applied.

Defenses Throughout the System Lifecycle

Systems have lifecycles, typically with the following phases: requirements analysis, design, implementation and testing, operation, emergency operation, routine maintenance, and reengineering or replacement with a new system. The phases are often treated as independent. For example, a vendor may develop a system, then turn it over to others to operate. The problem with such an approach is that lessons learned during operation, where defenses in depth are tested against real hazards, may not be incorporated into designs. Responses to hazards, new hazards, and interactions between defenses must be known by designers to avoid creating, or recreating, latent conditions. The following discussion will emphasize the importance of the design phase in NISs, both in avoiding latent conditions and reducing opportunities for active failures in operations and maintenance. Also important is feedback from the operations and maintenance phases to ensure that next-generation systems incorporate lessons learned in the field.

The design phase

Design philosophies. The fundamental attitudes and approaches of designers are critically important to NIS safety. What do the designers consider important? How do they approach their work? Consider two nuclear power plant control systems. The first system addresses only shutdown; it is independent of the normal control system. It consists of "about 6000 lines of code and uses only the simplest, most straightforward coding techniques" (Leveson 1994, p. 4). The second system attempts both normal control and shutdown and has 100,000 lines of code. All other things being equal, it is much easier to validate and trust the first system. The second system is more likely to contain latent conditions, including unknown connections between the two functions (control and shutdown). Thus the philosophy of the designers, even before they begin to design, can have a large effect on the latent conditions that will be introduced into the system.

Architectural patterns. In mature technologies, such as building design, knowledge of successful designs is stored and transmitted through architectural patterns. For example, curtain wall buildings did not always exist.[4] They were invented, and the pattern has spread worldwide. The HTTP protocol can be seen as a similarly successful pattern for NISs.[5]

Patterns are drawing significant attention in the software industry (e.g., Fowler 1997) because they help to familiarize developers, who may work in many different domains, with known problem-and-solutions sets. Patterns present the essence of a solution in such a way that the solution can be tailored by a developer. Most significantly, patterns emphasize important and unobvious aspects of problems. NIS patterns might, for example, show reliability vs. security tradeoffs and explicitly state decisions that must be made. For a developer with no experience in

[4] Traditionally, buildings were supported by their walls. In a curtain wall design, the building is supported by internal pillars, and the outside walls are simply facades: they hang like curtains. This approach would not be intuitive to an architect of the 1700s.
[5] Unfortunately, many NIS trustworthiness attributes such as security are "non-functional requirements" with significant costs (Schneider 1999, p. 71), so there is less impetus for their spread than attractive new features.

such tradeoffs, this could be of great assistance in avoiding bad assumptions that would lead to latent conditions.

Possible NIS patterns might include *high consequence intranet*, *firewall status board*, and *maintenance reminder board*.[6] The high consequence intranet pattern might emphasize the real consequences that have resulted from intranet failures and the commensurate investment needed in reliability. The status board would show overall system state-of-health in an understandable way. This could help to both restore state properly after maintenance and to understand state in novel events by showing a conceptual model of the system (Schneider 1999, p. 44). Innovative approaches to industrial process control displays (e.g., Vicente 1996) may be adaptable to NISs. The maintenance reminder board would help maintainers to remember actions and components.

Collaborative design. System designers must always make assumptions about how systems will be used. These assumptions are mature and realistic in many domains (e.g. electrical generators) but much more uncertain in complex systems like NISs that may be applied to a variety of domains (from Internet "chat" to military command and control). Tradeoffs are inevitable, and the only way to make good decisions about them is to cooperate with users. Collaborative design can capture specific hazards and needs learned from operational experience that may be completely unobvious to designers. Emergency operations such as attacks or component failures offer rich insights into both defenses and hazards.

For example, the "critical decision method" captures worst-case situations that experienced operators have faced in using a particular system (Klein 1998, p. 189). This method can provide insight into paradoxical defenses, such as what assistance can be given to operators and what can hinder their success. Thus collaborative design is particularly valuable in determining appropriate automation and how it should interface with human operators.

The operations phase

The incidence and effects of latent conditions can still be reduced in the operations phase. For example, standard operating procedures and rules that are developed and maintained by operational teams tend to be of better quality and are more likely to be followed than those simply passed down from management. Procedures like "plan, do, act, test" can be quite effective if they are enforced by peer pressure and not excessively burdensome (Reason 1997, p. 146). Checklists are a timeless and highly effective tool for reinforcing rules and avoiding slips.

Maintenance phase

The paradoxical nature of maintenance as both defense against, and possible source of, failure can be reduced even in systems with latent conditions. One key lesson from industrial experience is good reminders. Even the NIS equivalents of the "lock-out/tag-out" procedures used by electricians could prevent active failures. Checklists on clipboards and physical tags are still highly effective.

[6] It is interesting to note that the latter ideas reuse the pattern of an industrial control panel or board.

Both the operations and maintenance phases must have feedback loops to the design phases so that system evolution or reengineering reflects both lessons experienced and lessons transferred from others. Often these loops are difficult or impossible for operations and maintenance personnel to establish, so they must be established formally by management or instigated informally by designers.

Continuous Safety Management Challenges in NISs

The management of NISs is a moving target because market demands and technology are evolving quickly and unpredictably. Three factors will make the reduction of errors and accidents through safety a continuing challenge.

1. Immaturity of NIS technology and industry

Consider the following quotations on high consequence systems:

> "The safety features designed for the [systems] did not work as well as predicted because they were not based on scientific understanding of the causes of accidents" (Leveson 1994, p. 7)

> "Most designs for [systems] and safety features were based on the assumption that owners and operators would behave rationally, conscientiously, and capably. But operators and maintainers were poorly trained, and economic incentives existed to override the safety devices in order to get more work done. Owners and operators had little understanding of the workings of the [systems] and the limits of [their] operations." (Leveson 1994, p. 3)

These statements could be about NISs today, yet they describe the engineering and operation of high-pressure steam engines 150 years ago. It is clear that engineering, and society at large, goes through learning processes with new technologies. These processes, which lag behind demonstrable technology by years or decades, include increasingly refined perceptions of the costs, benefits, and risks of new technologies. Sometimes the learning process must be repeated when a false sense of security results from an absence of incidents or accidents (Reason 1997, p. 112).

2. Limited risk perception

Decisions are often made on the basis of risk perception. For instance, management may decide that survival and profit are higher risks than NIS reliability and allocate resources accordingly. While risk perception is sensitive to many economic and social factors, it has significant vulnerabilities and biases. In settings where consequences appear to be large, it may be prudent to perform quantitative risk assessments.

High-pressure steam engines were unprecedented technology. Analytical methods for dealing with gas dynamics were very limited, and risk assessment and system safety techniques were unheard of. In the case of NISs, we have extensive resources on which to draw. For example, it may be possible to transfer techniques and technologies from high consequence, high reliability systems such as nuclear weapons to NISs. These systems have traditionally been designed to achieve ***surety***, a near optimal combination of safety, security, and reliability (Kuswa 1998; Van Devender 1998).

3. Resource tradeoffs

Protection processes like those discussed in this paper compete with production processes for resources (Reason 1997, p. 4). Thus high level decisions made to optimize one can have impacts on the other. For example, an NIS service provider may invest in additional capacity rather than a surety assessment of their existing capacity. Again, this relates to perceived risks and opportunity costs. Even though an accident may have high consequences, more short term consequences (loss of market share, next-quarter profitability) may loom larger in manager's minds. As one NIS manager at a large bank said, "Money spent on protection is spent, but loss is probabilistic."[7] There appear to be fundamental conflicts between cost reduction and commoditization of NISs and our increasing reliance on them in high consequence domains.

Conclusions

Just like other large and complex systems before them, NISs are vulnerable to human errors such as slips, lapses, mistakes, and violations. These errors are most visible as active failures at the hands of front-line people, typically operators and maintainers. But such errors alone are often insufficient to cause accidents. Systems are too well protected by layers of defenses. But when less obvious errors have occurred, latent conditions may reside in the system. These combine with active failures to allow accident trajectories to penetrate the defenses.

Defenses can also be paradoxical; they may introduce their own hazards. Two paradoxical defenses are automation, which can surprise humans into making errors, and maintenance that contributes latent conditions.

Errors can be reduced by defenses in the system lifecycle, particularly during the design phase. Design philosophies (e.g., tendency to build big and complex vs. small and simple) can have a large effect on latent conditions and the chances for humans to successfully recover from accidents. Architectural patterns have considerable promise in helping designers to see hazards that may be unobvious and to reuse successful solutions. Collaborative design is particularly valuable because it allows designers to benefit from the real-life lessons learned by operators and maintainers. Often, these front-line individuals have unique insights into both external hazards and system behaviors.

As NISs and the demands placed on them evolve rapidly, they will produce novel – but also foreshadowed – errors and accidents. The "meta-systems" of many interacting NISs may be particularly challenging. Fortunately, there are techniques and approaches that can be transferred from system safety work in more mature domains. NISs do not have to be the exploding steam boilers of the early 2000s.

[7] Statement made by a panel member at the Cross-Industry Working Team (XIWT) Symposium on Information Infrastructure Robustness, held 3-4 Nov 1998, in Crystal City VA.

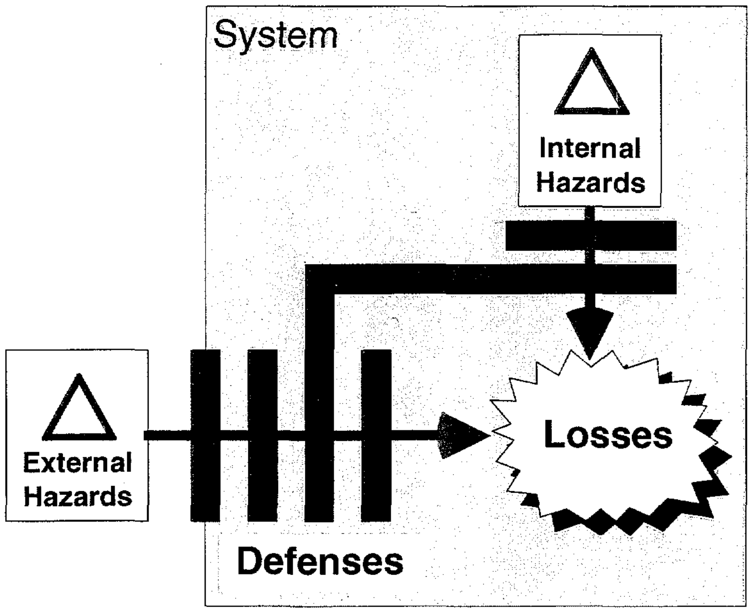

Figure 1: Hazards penetrate defenses to cause damage (adapted from Reason 1997, p. 3). In the case of NISs, hazards can be both external (e.g., malicious attacks) or internal (e.g., user mistakes). Defenses such as firewalls and file access control provide barriers to internal, external, or both types of hazards.

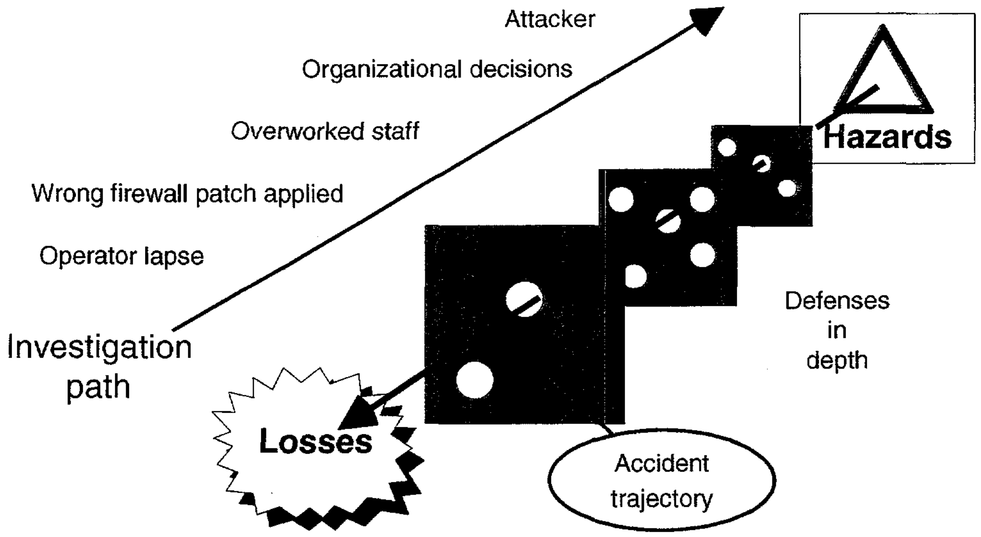

Figure 2: An NIS accident trajectory that penetrates holes in multiple defenses (after Reason 1997, p. 12). Some of the holes are from active failures, and others are from latent conditions. The investigation path works backwards along the trajectory to discover latent conditions that will likely permit other accident trajectories if not corrected.

References

Fowler, Martin. 1997. *Analysis Patterns.* Addison-Wesley.

Klein, Gary. 1998. *Sources of Power: How people make decisions.* MIT Press.

Kuswa, Glenn W. 1998. Corporate surety: A lifeline to survival. *Strategy & Leadership* (Strategic Leadership Forum) 20:5, November/December 1998.

Leveson, Nancy G. 1994. High-pressure steam engines and computer software. Presented as a keynote address at the International Conference on Software Engineering in Melbourne Australia, 1992 and published in *IEEE Computer*, October 1994. Available at http://www.cs.washington.edu/research/projects/safety/www/papers.html

Perrow, Charles. 1984. *Normal Accidents: Living with high-risk technologies.* Basic Books. [second edition from Princeton Univ. Press in 1999]

Reason, James. 1997. *Managing the risks of organizational accidents.* Ashgate Publishing.

Schneider, Fred B. (Editor). 1999. *Trust in Cyberspace.* Report of the Committee on Information Systems Trustworthiness of the National Research Council. Washington, D.C.: National Academy Press.

Slabodkin, Gregory. 1998. Software glitches leave Navy Smart Ship dead in the water. *Government Computer News*, 13 July 1998.

Van Devender, Pace. 1998. Sandia Surety Program White Paper. Available at: http://www.sandia.gov/Surety/SurWP.htm

Vicente, Kim J. 1996. Improving dynamic decision making in complex systems through ecological interface design: A research overview. *System Dynamics Review* 12:4, Winter 1996, pp. 251-279.

Distribution:

	0428	William Norris, 12300
	0445	Michael J. Skroch, 06232
	0449	Brad Wood, 6234
	0449	Judy Moore, 6234
	0451	Ron Trellue, 6238
	0451	Jennifer E. Nelson, 6235
	0455	Laura Gilliom, 6232
	0490	Perry D'Antonio, 12331
	0490	Richard Smith, 12302
	0490	John Covan, 12331
	0490	Arlin Cooper, 12331
	0490	Paul Werner, 12331
	0535	Larry Dalton, 2615
	0619	Review & Approval Desk, 15102 For DOE OSTI
	0630	Pace Van Devender, 4010
	0635	Glen Kuswa, 12365
	0638	Mike Blackledge, 12326
	0639	Robert Goetsch, 12303
	0741	Samuel G. Varnado, 6200
	0747	Alan Camp, 6412
	0766	Dori Ellis, 6300
	0769	Dennis Miyoshi, 5800
	0829	Kathleen Deigert, 12323
	0829	Chris Forsythe, 12323
	0830	Janet Sjulin, 12335
2	0899	Technical Library, 4916
	0977	Bill Cook, 6524
	1137	Kenneth E. Washington, 6534
59	1138	John H. Ganter, 6533
	1138	Larry J. Ellis, 6531
	1138	Bruce N. Malm, 6532
	1138	Sharon K. Chapa, 6533
	1138	Bryon Cloer, 6531
	1138	Craig Dean, 6531
	1140	James K. Rice, 6500
	1145	Ted Schmidt, 6430
	1202	W. Earl Boebert, 5901
	1207	Jim Yoder, 5909
	1425	Richard Cernosek, 1315
	1425	Robert Hughes, 1715
	9018	Central Technical Files, 8940-2