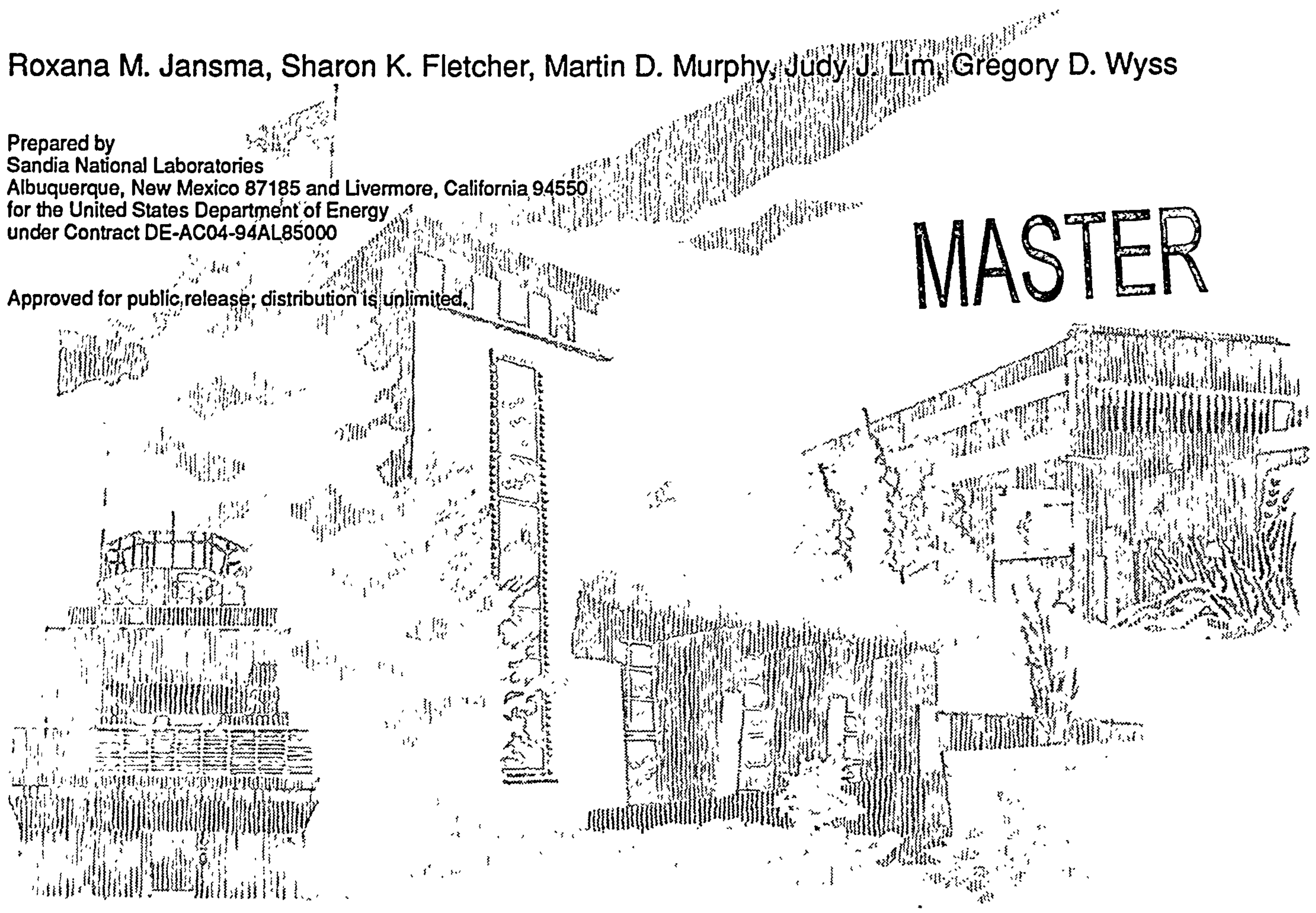
SANDIA REPORT
SAND96–2027 • UC–705
Unlimited Release
Printed July 1996

Risk-Based Assessment of the Surety of Information Systems

Roxana M. Jansma, Sharon K. Fletcher, Martin D. Murphy, Judy J. Lim, Gregory D. Wyss

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000



SF2900Q(8-81)

[illegible]

SAND96-2027
Unlimited Release
Printed July 1996

Distribution
Category UC-705

Risk-Based Assessment of the Surety of Information Systems

Roxana M. Jansma
Data Systems Security Department

Sharon K. Fletcher, Martin D. Murphy, and Judy J. Lim
Software Surety Department

Gregory D. Wyss
Risk Assessment and Systems Modeling Department

Sandia National Laboratories
Albuquerque, NM 87185

Abstract

When software is used in safety-critical, security-critical, or mission-critical situations, it is imperative to understand and manage the risks involved. A risk assessment methodology and toolset have been developed which are specific to software systems and address a broad range of risks including security, safety, and correct operation. A unique aspect of this methodology is the use of a modeling technique that captures interactions and tradeoffs among risk mitigators. This paper describes the concepts and components of the methodology and presents its application to example systems.

Acknowledgments

The authors thank Ronald Halbgewachs for his insights into system vulnerabilities, Paul Sands for his representation of a computer security viewpoint, and both of them for their valuable contributions in other areas of the project. Special appreciation goes to Peter Watterberg, who originated the approach developed in this project and provided leadership and motivation in the important early stages of the work.

Contents

Figures

Table

Risk-Based Assessment of the Surety of Information Systems

1 INTRODUCTION

During recent years there has been a dramatic increase in the use and importance of automated information systems in broad aspects of life. Along with this increased use have come system failures causing the loss of money and compromise of information, and potentially endangering human life. The importance of ensuring correct system operation and understanding the risks of failures is increasing as well. After all, the overarching goal is to produce systems that do what they are supposed to do and not what they aren't supposed to do; that is, build the right thing, build it well, and protect it appropriately. To reach such a goal requires a dynamic, whole system, whole lifecycle perspective.

At Sandia National Laboratories, the term "information system surety" has been coined to convey this idea. The term "surety" is borrowed from the realm of nuclear weapons, where it refers to safety, security, and use control. Information system surety has a broadly analogous meaning and can be defined as "ensuring the correct operation of an information system through the integration of appropriate levels of access control, integrity, utility, availability, and safety." The focus is on integrating and balancing surety domains or objectives that are traditionally treated separately. The surety objectives are briefly defined as:

- Access control -- confidentiality, privacy; control of knowledge of information or use of resources
- Integrity -- completeness, validity, authenticity
- Utility -- fitness for a purpose, correctness
- Availability -- accessible and usable when and where needed
- Safety -- freedom from harm to persons or property

The key to surety in an information system is identifying and mitigating the risks of system failure. Such a goal requires a balance of potentially conflicting surety objectives. For example, availability requirements may compete with access control requirements. Risk is never zero; residual risk must be understood before its importance can be judged and a decision can be made regarding what, if anything, to do about it. An understanding of system risk will also guide assessments of how available technologies may provide increased system surety.

A technique is needed that aids in identifying the most important surety issues given the specific use for which a system is intended. Sources of risk must be determined. Potential consequences resulting from system failure must be identified and prioritized. Risk mitigation techniques must be analyzed to determine not only their effectiveness at reducing risk but also any interactions or side effects. Only then can decisions be supported regarding how to mitigate risk and whether the remaining risk is acceptable.

An interdisciplinary team at Sandia National Laboratories developed a risk-based quantitative methodology for assessing the surety of information systems. The effort was funded through the Laboratory Directed Research and Development Program. The methodology includes not only a risk-based surety evaluation modeling method but also a framework to guide and support model development.

Clearly, many gaps exist in current knowledge of how to manage and assess risks in software systems. The work described in this report addresses the development of a coherent, whole-system framework for reasoning about information system risks at any desired level of detail. This work is innovative in many ways. It presents a unique perspective and systematic framework for evaluating system surety problems and solutions. It transcends the traditional security approach which is rooted in compliance with dictated solutions. It broadens the scope to encompass access controls, integrity, availability, utility, and safety. It expands the view to include the operational system, its development and maintenance, its architecture and interfaces, and state changes. It allows an exploratory, iterative, risk analysis completely tailored to the system at hand. It transcends current mathematical risk analysis approaches in order to provide residual risk information in a form that is useful for improving the system surety and for refining the analysis. It emphasizes appropriate mathematics and uncertainty analysis to ensure that risk calculations are meaningful. This work brings the effectiveness and formality of the Probabilistic Risk Assessment field to bear on the problem of software system surety in a rich and powerful way, which opens the door for new development and assurance paradigms based on quantitative risk mitigation.

2 VIEWS OF RISK IN SOFTWARE SYSTEMS

For purposes of discussion, we view the progression of thinking about risk management and assurance in terms of three "generations." These descriptions are meant to capture the prevalent views of the times, although there certainly might have been pioneers who were ahead of the times. Each generation represents a major paradigm shift in the community - a different take on how to view the problem and its solutions.

2.1 The First Generation (Rated Systems)

The first generation of software system risk management was security- and compliance-oriented, and required buy-in to a predefined set of risks which was assumed to apply to all systems. Risk concerns revolved around certain aspects of "CIA" (confidentiality, integrity, and availability). These views evolved in an environment characterized by mainframe computers and protection of classified information.

Not only was the set of risks fixed, but the mitigation strategies were dictated. These strategies included access controls at the system and file levels, encryption for network transmissions, and disaster recovery planning. Several levels of systems were defined, where a higher level meant that more of this security model was implemented and/or it was implemented with more rigor for correctness.

In this generation, risk assessment was, for the most part, missing. Of course, some form of initial risk assessment occurred which defined the CIA set of risks for all systems. Beyond that, system-specific risk assessment usually included only site-specific disaster recovery concerns, and determining which security level applied. There was little leeway for customized solutions, never mind optimal solutions. A system either did (compliant) or did not (non-compliant) implement the required strategies.

While restrictive, this approach succeeded in its environment. The first generation made assurance straightforward for the consumer: buy rated products. It was also straightforward for the vendor: get rated. The picture was compliance-oriented. A major problem being encountered, however, is the difficulty of composing rated products into today's more complex systems. If it can even be done, it may result in overkill solutions at unacceptably high cost, or may produce a system of rated components that has severe flaws.

2.2 The Second Generation (Protecting Assets)

The advent of, first, networks, and then, distributed processing on those networks, was very problematic for the first generation risk mitigation approach. The techniques that had been adopted did not easily extend into these more modern environments. Also at work was concern that the first generation CIA risk model simply did not fit all applications. A need was felt for system-specific risk assessment. Other fields, such as nuclear power, weapons, and aviation, were taking a systems view and using analytical risk analyses; their success provided encouragement for a risk-assessment approach. As a result, a new view of risk has emerged for software systems, and the National Institute of Standards and Technology (NIST) played a prominent role in prompting this view to coalesce [NIST, 1991]. This new view is based on the following system components:

- vulnerabilities
- threats: active, passive
- assets: data, hardware, software
- impacts: disclosure, destruction, modification, unavailability
- types of mitigation: avoid, transfer, reduce threat, reduce vulnerability, reduce impact, detect & respond, recover

This emerging view of risk is expressed: "A threat is realized through a vulnerability, which impacts an asset." This more general view of risk, to be applied on a system-by-system basis, represents a major advance.

Independent of the NIST model, Parker [Parker, 1991] has expanded the above assets list by breaking software into applications and operating systems, and adding users, renaming the list "levels of abstraction". He has also evolved CIA into a list of "security attributes" consisting of confidentiality, authenticity, integrity, utility, availability, and possession. These two lists form the axes of a matrix for exploring risks and countermeasures related to protecting assets.

Little progress seems to have been made beyond these definitions, though, and this is due to two major roadblocks. First, the software community doesn't know how to measure the risk mitigation achieved by a design, and thus, how to draw any conclusions about assurance. Second, the community lacks a coherent framework for assessing tradeoffs among various concerns and the interactions among their mitigators.

But developers are increasingly rejecting the underlying mindset that the first generation approach of complying with orders actually provides the needed surety. The second generation risk model represents a positive trend toward assessing a system's actual surety needs. Unfortunately, no one has yet proposed an assurance technique for this model. The Federal Criteria [Fed. Crit., 1992] and Common Criteria [Common Crit., 1994] may provide a step in the direction of customized risk mitigation, but it's still within the rated product approach. Many emerging software development techniques which contribute to higher integrity systems are not addressed by either the second generation risk model or the Federal Criteria. As long as there is not an assurance technique that credits good practices, developers will unfortunately sacrifice doing things right in order to apply scarce resources to doing things that are measured.

Risk assessment tools for software systems have appeared on the market in recent years. However, many of these are simply computerized checklists which measure compliance with the first generation's prescribed risk mitigators. Others are more aligned with the second generation view; however, many of these take the assets-protection viewpoint to an extreme and focus on computing Annualized Loss Expectancy, converting all assets to dollar equivalents. Tools are still badly needed that help quantify risk mitigation in a useful manner, and that can deal with the interactions among risks and mitigators.

2.3 The Third Generation (Managing Risk)

We believe the software community is standing on the brink of a third generation. While the viewpoint has shifted from rated systems to protecting assets, one more shift is needed, to managing risk. The third generation of risk management will be characterized by:

- A fundamental change of perspective to one which more fully facilitates total risk management
- Emphasis on correct system operation through appropriate levels of utility, access control, integrity, availability, and safety
- Consideration of actual threats, inherent vulnerabilities, and the feasibility and cost/benefit of safeguards as the basis for making system decisions.

This generation will have a dynamic, whole system, whole lifecycle perspective: build the right thing, build it well, and protect it appropriately. This is what we really care about.

Providing a viable framework is key; we must have a useful underlying perspective on risk assessment and risk management within which to work. It is imperative that enough effort be devoted to deriving a good framework, for this is the foundation, the view into the problem space, that colors how we are able to see solutions. For example, while a narrow view of sabotage might focus on virus protection in an operational system, a whole lifecycle view will encompass protection throughout design, implementation, delivery, and maintenance. And while a narrow view of network security might focus on encrypting communications, a whole system view will explore whether network nodes have compatible security policies and whether they exchange sufficient security information to uphold the policies. And while a narrow view of integrity might address mechanisms within a properly operating database, a dynamic view will also look at shutdown-startup synchronization issues. The work described in this report focuses on developing a third generation framework, and a methodology for achieving risk management within this framework.

Some parts of the community have already adopted a risk management mindset, especially where software safety is concerned. They are using risk reduction as a basis for design decisions, sometimes following fault or event tree types of thought processes to identify risks, and documenting each of the chosen risk mitigators along with a qualitative estimate of the mitigation it achieves. This is an excellent start. We are addressing these advances: that safety and security and dependability must and can be considered together within a single context; that each risk and its mitigators need not be addressed in isolation from others (which ignores the reality of serious interactions); that we can begin to quantify; and that better guidance, specific for software systems, can be provided to the art of the analysis. These advances are non-trivial, but are necessary for a true third generation environment.

Availability of third generation tools means that software system design and redesign can be carried out with a full understanding of how security and other risks have been mitigated, and with conscious acceptance of remaining risk. It also provides a documentation of original surety requirements and design choices, and allows future system enhancements and changes to be evaluated for their effect (either positive or negative) on system surety. It facilitates a risk analysis completely tailored to the system at hand, instantiating its threats, its barriers to those threats, its needs for risk reduction, and the interactions among all these elements. The previous generations' views of software system risk management have been too narrow in scope to support development of such comprehensive tools.

3 CURRENT APPROACHES TO RISK ASSESSMENT FOR INFORMATION SYSTEMS

Researchers and practitioners currently disagree about the usefulness of existing and proposed techniques for assessing risk. The underlying issue is the problem of predicting the surety of a software-based system, and the effectiveness of a method for obtaining surety.

3.1 The Case for Probabilistic Risk Assessment

We believe that an integrated systems methodology entailing probabilistic risk assessment (PRA) offers the best means for addressing the problems in software surety. PRA must be an intrinsic part of any competent process for producing software systems for several reasons.

First, software is often an essential part of systems for which a probabilistic requirement may be stated (e.g., the probability of a very serious accident attributed to the system must be less than some acceptable threshold), and stating that the designers used good engineering practices to preclude serious accidents does not tell us how well they achieved their objective. Accepting a system on this basis implies a high level of trust that these practices always yield systems that are trustworthy to the desired level. Such trust is unwarranted for real, complex software systems since various accidents (some catastrophic) have occurred.

Second, an analysis that quantifies the attainment of some surety level requires the explicit statement of the probabilistic model used and hence the underlying assumptions about which events are credible, the cause-effect relationships among system components, the sensitivity of the model to variations in the parameter values, etc. The development of the model forces one to recognize what is not well understood about the system and problem environment, especially when one makes simplifying assumptions. This type of rigorous model can be scrutinized for errors and unwarranted assumptions and exercised to determine the effect of different assumptions.

Third, engineering for surety is a matter of choosing the right compromise and selecting between alternative designs which include different compromises. The only rational way to choose the best among alternative designs is to model the effects of the design candidates on system surety and compare them. The model makes explicit the decision criteria, right or wrong, for design selection.

In the following sections we present brief evaluations of common PRA methods such as fault tree and event tree analyses. We found that none of the common methods were appropriate to *all* aspects of the information system surety problem. We also found that, while the thought process for evaluating most aspects of surety began by looking for ways that the system's proper "process flow" could be disrupted, diverted, or caused to exhibit undesired behavior, most of the common methods did not support such a modeling approach. For these reasons, we examined more generalized directed graph techniques and found that influence diagrams provided a good point of departure for our risk assessment studies.

3.2 Analysis Techniques: Relative Strengths and Limitations

Certain analysis techniques have proved durable because they have been able to represent and be used to examine important classes of problems and types of systems. We shall discuss some of these techniques, starting with the premise that the problem attributes and system characteristics determine which technique would be most applicable or powerful for any given situation. The reader is assumed to possess a fundamental knowledge of the basic model structure and mathematics underlying each technique. Interested readers may refer to the ample textbooks and sources in the literature for discussion of such basics. Here we will focus on the relative strengths and limitations of the techniques experienced thus far.

First we discuss several techniques created for analyzing the reliability and safety of physical systems: failure modes and effects analysis, reliability block diagrams, fault trees, and event trees. These techniques involve models of cause and effect. Their extension to software systems poses a challenge and requires definition of rules for problem decomposition. However, these techniques are well understood and have proven useful in other industries.

Failure Modes and Effects Analysis (FMEA). FMEA and its extension to failure modes, effects, and criticality analysis (FMECA) are the most elementary of the techniques. They are used to analyze the consequences of component failures and are largely qualitative analysis procedures, consisting of tables constructed for the system components and the possible failure modes of each component. Typical information elicited in a FMEA or FMECA include component description, failure mode, effect of failure, cause of failure, occurrence, severity, probability of detection, risk priority, and existing or proposed corrective actions. FMECA then adds a formal criticality analysis to rank the results. Both FMEA and FMECA are subject to the same strengths and limitations as event trees, and tend to be very time consuming and hence expensive to perform. Moreover, FMEA and FMECA lack a model infrastructure for integrating the component information collected in the various tables. Finally, doing a FMEA or FMECA on software may not be realistic because of the extremely large number of possible software outputs and behaviors that would usually have to be considered.

Reliability Block Diagrams. In a reliability block diagram, blocks represent system components. They are connected together to represent failure dependencies. If the failure of any of a set of components will cause the system to fail, a series connection is appropriate. If the system will fail only if all components fail, a parallel connection is appropriate. More complex topologies are also possible since failures in real systems are usually more complex than can be represented by building diagrams from series and parallel parts. Analysis of a reliability block diagram in the most general case is complicated. (Complex reliability block diagrams can also be transformed into fault trees and solved by standard fault tree solution techniques.) The advantages of this model are its simplicity as a visualization tool (akin to a system block diagram) and straight-forward solution when the model can be constructed from series and parallel components. Limitations of the model are the assumptions that component failures are statistically independent and that failures are constant over time.

Fault Tree Analysis (FTA). FTA is a deductive hazard analysis technique which considers a system failure and then provides a top-down approach to reason about the system or component states that contribute to the system failure. FTA starts with the definition of a particular undesirable event as the "top event" of the tree. The system is then analyzed to determine all the likely ways in which the undesired event could occur, and the fault tree is developed by successively breaking down events into lower-level events that generate the upper-level event. Hence the fault tree model is a logical representation of the various combinations of events that lead to the undesired event. The faults may be caused by component failures, human error, environmental conditions, or any other event that leads to the undesired event. It should be noted that a fault tree is not a model of the system or even a model of the ways in which the system could fail. Since a fault tree is comprised of two elements, logic gates and events, a fault tree is thus a graph of the logical interrelationships of basic events that may lead to the "top event." As for the strengths of the technique, FTA provides a systematic framework for keeping a problem tractable in that the model is only concerned with failures that lead to a particular undesired event. FTA also helps the analyst to be complete "categorically" in examining the various failures. On the other hand, fault trees do not handle dynamic or time-dependent events very well and may not reveal consequences of events occurring in the middle of the tree. Also it should be noted that FTA assumes that basic events are mutually independent. Thus if the fault tree model fails to explicitly account for potential common mode and common cause failure mechanisms, the fault tree solution can yield an unrealistically optimistic (low) top event probability.

FTA has been extended to software systems where fault trees may be built for a given system based on the source code for that system [Leveson and Harvey, 1983]. The analysis starts at the point in the code that yields the potentially undesirable outputs. The code is then analyzed in a backwards manner by deducing how the program could have gotten to that point with the set of values producing the undesirable output.

Event Tree Analysis (ETA). ETA is an inductive hazard analysis technique that considers a specific fault in some component of the system and examines in a sequential manner what the consequences of that fault will be. The approach taken is to consider an initiating event and its possible consequences, then for each of these subsequent events, the potential consequences are considered. In essence, ETA is forward thinking and considers potential future problems. The probabilities of the final outcomes are obtained by multiplying the probabilities of the events comprising the path. For systems for which little is known, event trees can be very useful for analyzing consequences of individual components to determine if a mishap might occur and what that mishap might be. ETA can help determine whether single points of failure exist for the system. At the same time, the advantage afforded by event trees can also be a weakness of the technique. The initiating events for ETA may be both desirable and undesirable since a desirable event could possibly lead to an undesirable outcome. This means that the set of initiating events is the entire spectrum of events that may occur in the system. Thus much analysis time may be wasted by considering an event tree from a given event, such as the failure of a sensor, when that event may never lead to a mishap. It should also be noted that ETA, like FTA, assumes that event probabilities are independent of one another. Thus, one must explicitly account for the interdependencies between events in the structure of the event tree model itself in order to obtain realistic ETA results.

Analysis techniques that provide more refined ways of handling system dynamics and dependencies will be discussed next.

Petri Nets. A Petri net is a graph-theoretic model that can express concurrency and asynchronous behavior as information or control flows. The major use of Petri nets has been the modeling of systems of events where some events occur concurrently, but there are constraints on the concurrence, precedence, or frequency of these occurrences. A Petri net consists of circles (called places) representing conditions and bars (called transitions) representing events. The places and transitions are connected by directed arcs. The execution of a Petri net is controlled by the position and movement of markers (called tokens) in the net. The tokens are subject to certain rules and are moved by the firing of the transitions of the net. A transition is enabled for firing only if there is at least one token in each of its input places. The transition fires by removing one token from each input place and placing a new token in each output place. Given this model construct, Petri nets prove to be very useful in modeling or simulating system state changes caused by triggering events. The benefits of Petri nets are that they are intuitive, easy to understand through visualization, and can analyze small systems comprehensively. However for larger systems (and not necessarily complex ones) Petri nets lose their scrutability and the system properties become obscured in the graph. The analysis results show the presence or absence of system operational properties such as hazardous conditions, system deadlock, or unreachable states and their associated probabilities. Complete path or "scenario" information is not a natural output of the model.

Markov Models. Markov models are directed graphs that capture the concepts of system states and probabilistic transitions between states. They provide a natural, direct representation, through the use of cycles, of systems whose components are repairable and systems where component failures have interactions. Recall that fault trees and event trees are acyclic graphs and hence do not readily accommodate these system characteristics. The two basic forms of Markov models are chains and processes. A Markov chain uses matrix multiplication in discrete time to obtain the state transition probabilities; a Markov process uses a set of differential equations over continuous time. Relative to the other techniques discussed, Markov processes require a more sophisticated knowledge of mathematics for their solution. In fact most Markov models of real systems have many states and hence are difficult to solve, requiring simulation. Again, complete path or "scenario" information is not a natural output of the model.

Formal Methods. This term refers to techniques that have a sound basis in mathematics and employ an associated mathematically-defined notation. For instance, most formal methods have set theory and predicate logic as their underlying basis. Formal methods have been used in two distinct ways: first, for the production of specifications used as the basis for conventional system development; and second for the development of specifications which are then used as the basis against which the correctness of the program is verified. In the first case, the mathematics is used as a documentation medium that offers the benefit of precision, eliminating the risk of misinterpretation of the specifications, plus the opportunity to postulate and "prove" certain desired properties of the specification. In the second case, an additional benefit is provided in which it is possible to show that the program does what it is specified to do with the same degree of certainty as a mathematical proof. However, we also note that proof of correctness is not a trivial task and few practicing software engineers have the necessary skills to use formal methods.

However the fundamental limitation remains the problem of specification validation; that is, demonstrating that the requirement specification does not allow executions which would lead to catastrophic failure in the system's operational context. The real difficulty is that there is no way of knowing whether all of the threats and system failure modes have been identified, so one can never be sure of the completeness of the specifications. Thus the formalism and mathematics, of themselves, are insufficient to guarantee surety.

From our perspective, the practical benefit from using formal methods is their impact on the system developers' thinking process. The intellectual exercise increases knowledge of the software and hence increases confidence in the software. However, there are questions regarding the extent to which formal methods should be used and how one actually applies formal methods in practice. Some suggestions for the use of formal methods in the software development lifecycle of critical systems are made by researchers who recognize that formal methods today are fraught with limitations reflecting the immaturity of the techniques themselves and inadequacies of the support tools [Barroca and McDermid, 1992].

Other Methods. Techniques such as reliability growth models, software fault tolerance techniques, and software engineering tools are not discussed here since these methods have been more widely applied in software system surety than the analysis techniques we covered. The relative strengths and weaknesses of these other methods are already well described in the literature.

3.3 Surety Evaluation Modeling Methodology

Each of the common analysis techniques described above has its own strengths and limitations. No single technique can do it all. Analysts should not use modeling tools blindly but must carefully examine each one and understand how it works and what its limitations are. Rather than justifying one's preferred technique, we should investigate how the different techniques can positively reinforce each other. It is the combination of techniques that will lead to better systems, and we need more experience, examples, and applications of techniques to understand the limits to which information surety can be assessed.

In searching for good tools to help in risk-based design or risk management of software-based systems, we noticed an obvious need for improvement in the areas of model building, flexibility, iterative refinement, and vulnerability analysis. We believe that a PRA-based tool is best suited to our purposes. To this end, we have adapted a graphical technique, the influence diagram, to perform quantitative assessments of information system surety problems. The influence diagram formalism is conceptually similar to event tree and decision tree formalisms, and it has the advantage of supporting both backward and forward construction of the model. Conventional influence diagrams do not, however provide everything we consider important for analyzing information systems. Therefore, we propose some extensions in notation and a new solution methodology. We describe influence diagrams and our modifications in detail in section 4.5.

4 METHODOLOGY

In this section we describe our methodology for identifying sources of risk and risk mitigators and for creating the system risk model.

4.1 Overview

Traditional approaches to the assessment of information systems have often been based on an *ad hoc* or piecemeal approach in which individual requirements are generated to protect against various real or perceived threats. These requirements often relate only to a particular area of information system surety (availability, security, etc.), but their *combined* impact is rarely considered systematically. In addition, current analysis techniques are limited in the type of flaws they detect and/or in that they require too much time to perform to be feasible. Assessment of information systems under a total systems approach would help reduce these problems. We believe that an information system surety assessment approach should have the following characteristics:

- The approach should not be merely "checklist" or compliance-based, but should assess the consequences that would occur should the system fail to achieve its surety objectives (*appropriate* levels of functionality, security, etc.), and how much a user is willing to spend to avert those consequences.
- The approach should provide quantitative information with which tradeoffs between various design alternatives can be objectively evaluated.

- The approach should be readily extendible and encourage iterative refinement (i.e., it should support both "quick-look" studies and the extension of those studies to arbitrarily greater levels of detail as appropriate).
- The approach should provide guidance to help the analyst ensure that the model considers all appropriate threats, mitigation strategies, and consequences, and a quantitative screening technique to help the analyst decide which scenarios can be legitimately neglected.
- The approach should be easy to use, but powerful enough to cross the boundaries that currently separate the various domains encompassed by information system surety.
- The approach should be supported by software that would facilitate model development and automate model solution.

To facilitate system-specific total risk assessment and management, a methodology must provide the system analyst or developer with:
- assistance in identifying system risks, and therefore surety requirements
- assistance in selecting mitigation techniques (i.e., ways to avoid, reduce, transfer, or control sources of risk)
- assistance in understanding and quantifying the effectiveness, dependencies, and interactions of mitigators (the combined impact of each design alternative on correct system functioning vis a vis surety objectives AND functional aspects of the information system)
- the ability to tailor the surety model to the specific system

Our framework for assessing and managing risk in information systems provides this assistance through supporting components -- a Risk Identification Matrix and a Risk Mitigators Matrix. Our system risk model provides a graphical depiction of potential system states and the barriers that can affect the probabilities of state transitions. But, more than that, it is also the formal description over which risk calculations can be defined. Thus, it is the heart of the analysis method. However, software system designers and analysts might not be familiar with this kind of modeling, and might have difficulty constructing meaningful and complete models. Therefore, the two matrices were introduced as an aid to the model building. The matrices are intended to guide the analyst in determining surety requirements, areas of highest consequence, and appropriate technologies or approaches to address risks. These preliminary matrices provide support for development of a system risk model which can be analyzed to aid in system design decisions regarding risk mitigation and acceptance.

The components are used in a risk assessment process which involves:
- Building a system risk model, using input from the matrices
- Analyzing risk mitigators
- Running the analysis engine
- Evaluating remaining risk and refining the analysis through iterations

In our description of the process, the analyst is that person who is carrying out an analysis, whether it be on paper or using the software tools. In reality, many people may be involved in various aspects of an analysis - supplying individual expert opinion, reaching consensus as a team, supplying input, interpreting output, etc. We simply represent their total interaction with the process by the term "analyst".

The overall process, shown in Figure 1, is for an analyst to build a graphical model of system risks, using the risk identification and risk mitigators matrices as guides and sources of information. The analyst quantifies the model by performing a barrier analysis for each barrier (mitigator) that is to be considered, and a threat analysis for each threat to be considered. Then an analysis engine is run to compute remaining risk in the system. The job of the analysis engine is to perform appropriate computations on all quantified input from the analyst, and to return information on weaknesses in the system. If cost information is incorporated, then the analysis engine may return cost/benefit information as well. The remaining risk is evaluated and if it is unacceptable, modifications are made and the process is reiterated. The initial model should be at a high conceptual level and address only the highest risks. The analyst should then iteratively refine the model and re-run the analysis until useful levels of detail are achieved throughout the model, and all relevant risks are incorporated. The result of this process is a risk assessment and a risk management strategy for the system.

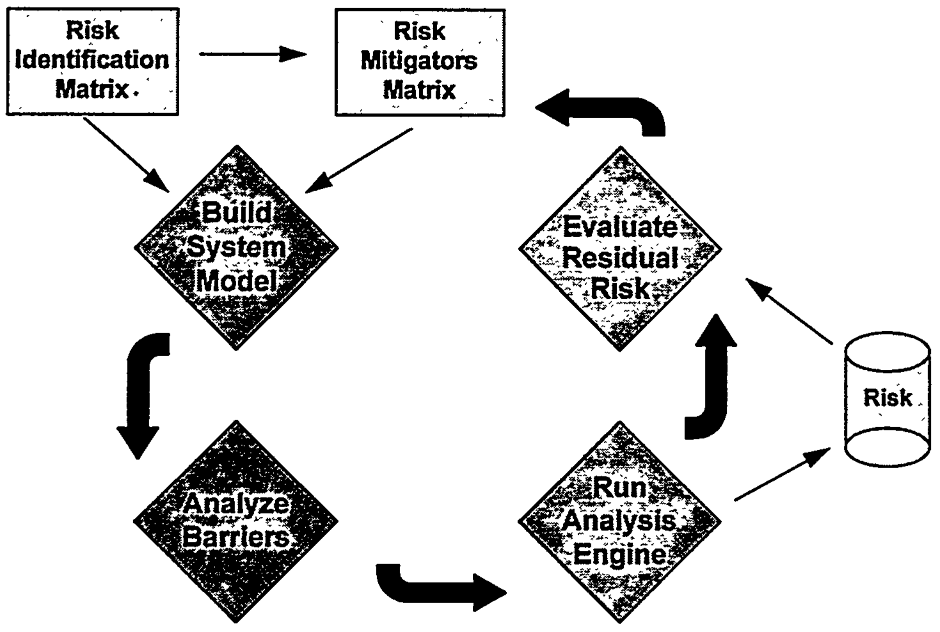


Figure 1. Risk Assessment Process

4.2 Supporting Components

4.2.1 Risk Identification Matrix

The Risk Identification Matrix (Figure 2) provides a framework for the analyst to determine the most important risks to the system based on

- the surety objectives for that system (rows in the matrix),
- aspects of the system that may give rise to risks (columns in the matrix), and
- consequence areas of particular concern (not shown in Figure 2).

System Aspects / Surety Objectives	Information	Processes/ Transactions	System Composition	State Changes	Interfaces
Access Control	• eavesdropping • access control failure	• authent. failure • system break-in • spoofing	• OS lacks features • passwords exposed on net	• access by repair personnel • abnormal event	• different surety policies across interfaces
Integrity	• input error • unauthor. mod. • process error	• repudiation • process diverted • subversion	• untrained users • DBMS lacks integrity check	• incomplete updates due to repair time	• bad input source • incomplete input
Utility	• inappropriate output • unauth. mod. • *accidental mod.*	• *inappropriate process* • unplanned environment	• *multiple copies on servers poses update problem*	•shutdown-startup not synchronized • *service prevents updates*	• *inappropriate input source* • output misused
Availability	• *access time too long* • inappropriate access control	• *system overload* • timing design fault	•unreliable comm. • *single points of failure* • sabotage	• *in maintenance* • sabotage • natural disaster	• *power interruption* • network service unavailable
Safety	• *incorrect info.* • insufficient info. • inaccurate info.	• *unchecked input or output* • out of tolerance	• design not fail-safe • *unclear procs.*	• *operation during abnormal environment*	• *unchecked input*

Figure 2. Risk Identification Matrix Example: Medical Decision Support System

The matrix is used to determine surety requirements in terms of perceived risk and desired risk reduction, in the context of potential consequences and their relative importance. The cells of the matrix contain sources of risk based on the surety objectives, system aspects, and consequences that should be considered. The intent is for each cell to contain the relevant sources of risk for any system, arranged as pieces of a taxonomy.

Surety Objectives. In devising a list of surety objectives that is meaningful for information systems, our intention is to encourage consideration of the whole system and its environment. The traditional security categories of confidentiality, integrity, and availability are not sufficient for this purpose, and we also felt that Parker's categories [Parker, 1991] lacked emphasis on correctness and safety. Our categories are not meant to be exclusive and non-overlapping, but are intended to encourage complete treatment and expanded thinking about system surety. We chose to address five surety objectives.

- **Access control** encompasses the objectives of confidentiality, privacy, and control of knowledge of information. It also deals with use of resources in addition to information/data, such as CPU cycles.
- **Integrity** addresses completeness, validity, and authenticity, not only of data but also such aspects as processes and system operation.
- **Utility** is defined as fitness for a purpose and correctness. It encompasses correct operation, of the system, data, processes, etc., to achieve the purpose for which the system is intended.
- **Availability** implies that the system, data, processes, etc. are accessible and usable when and where needed.
- **Safety** can be broadly defined as freedom from harm to persons or property caused by the information system. Some argue that this category is superfluous because safety will automatically be achieved when the other surety objectives are met. However, we feel that there are issues beyond simply meeting individual surety objectives and that safety merits additional emphasis.

System Aspects. The system aspects that we have chosen to consider deviate from the typical taxonomies like "information, software, hardware, network, and users". To address the goal of "correct operation of the system," we want to consider broader system aspects.

- **Information** or data plays an essential part in system operation.
- **Processes or transactions** performed by the system can include concerns such as format, completeness, timing, and guaranteed delivery.
- **System composition** includes design, architecture, operator/user, networking, platform, and interoperability issues.
- **System states** include not only normal operations, but also maintenance, shutdown, and abnormal or unplanned events, and especially, transitions between states.
- Consideration of **interfaces** encourages a look at how the system actually functions in context.

Consequence Areas. Different systems embody different concerns about consequences of failures to meet surety objectives. This taxonomy of consequence areas can help to identify and clarify surety requirements and to focus on areas of greatest importance or impact.

- **Mission** related consequences are those that affect the functionality of the system and its ability to fulfill its intended purpose.
- **Legal/Regulatory** consequences may be important where abiding by rules of outside agencies is an issue.
- **Worker Health and Safety** issues may arise when an information system is used to control potentially physically harmful elements.
- **Public Health and Safety** also deals with physical harm, particularly when it may not be contained within the area of the business.
- **Environmental** impact may result from problems with the system.
- **Political/Social** consequences may be paramount, even if no "actual" harm is done and mission requirements are still being met (e.g., public *perception* of safety issues).

Using the Risk Identification Matrix. The Risk Identification Matrix provides a framework within which the analyst can define system risks. The row and column labels, i.e., the "Surety Objectives" and the "System Aspects", assist the analyst to think through many different areas of concern. Off-line, the analyst should use whatever views of the system seem useful. For example, a physical view might be useful for thinking about risks arising from the system composition. A process flow view might be useful for thinking about safety risks. The risks captured in the matrix result from the analyst asking what could go wrong from these various views, and should be stated in terms of an "undesirable state or event" in the system. If the analyst can further identify states that might precede or follow the risk state, then he should begin to draw a graph around it. Severity of the risk in each of the "Consequence" categories should then be estimated on some scale, such as Low, Medium, or High.

The analyst populates each cell with the relevant sources of risk for the system. Ideally, if the matrix were fully populated with possible risks, the analyst's job would be one of pruning. Realistically, it must be one of pruning combined with supplementing.

The matrix is read:
"There is a [surety objective] risk relative to [system aspect] due to [risk source]."

Examples:
- "There is an [access control] risk relative to [system composition] due to [passwords exposed on the network]."
- "There is an [integrity] risk relative to [information] due to [process error]."
- "There is a [utility] risk relative to [state changes] due to [shutdown-startup not synchronized]."
- "There is an [availability] risk relative to [processes] due to [system overload]."
- "There is a [safety] risk relative to [interfaces] due to [unchecked input]."

Although the traditional impacts and assets are accommodated within this framework, it is much broader, giving rise to exploration of system dynamics (state changes), architecture choices (composition), and correct operation. For a particular system under analysis, the risk matrix will be pruned by the analyst to contain and prioritize only those risks of sufficient consequence and likelihood that they need to be mitigated.

The Risk Identification Matrix in Figure 2 shows some potential sources of risk for an example system; in this case, a decision-support system where a physician uses a distributed database system to find information on which to base treatment decisions. The analyst has determined that the primary surety objectives are:
- utility -- the system must fit the intended purpose and perform correctly,
- availability -- the system must be available when the physician wishes to use it, and access time must meet certain criteria, and
- safety -- the system must not function in such a way as to cause a decision that puts a patient's health at risk.

In this example, the analyst determines that the sources of risk that are of greatest concern are those shown in italics in the matrix. For instance, utility is greatly affected by accidental modification of data, inappropriate processes and input, and update problems.

Ideally, a "universal" Risk Identification Matrix could be created to provide a taxonomy encompassing all potential sources of risk for information systems. The analyst would not have to start from scratch, but could choose applicable risk sources of interest. The matrix and its underlying repository could be developed and updated by domain experts so that users of the methodology might leverage their efforts.

4.2.2 Risk Mitigators Matrix

The Risk Mitigators Matrix has the same format as the risk matrix, but the cells contain mitigators, or barriers, corresponding to sources of risk. The intent is that the mitigators not be limited to hardware and software technologies, but include rules and procedures, design and development practices, and cover the lifecycle spectrum. Thus credit can be given for using a proven real-time design architecture, for using a highly-rated software development methodology, for a trusted path delivery mechanism, for fail-safe design, etc. The effectiveness of the mitigators can be evaluated in terms of several characteristics, such as the degree to which technology vs. rules and procedures are involved, perceived strength, cost to implement, ease of use, outside dependencies, etc. The analyst selects, and/or adds, mitigators to try out in the analysis. The matrix may include nominal figure-of-merit values for those mitigators it does contain.

	Information	**Processes/ Transactions**	**System Composition**	**State Changes**	**Interfaces**
Access Control					
Integrity					
Utility	•visual scan •overwrite check •"diff"	•use a tested and proven process •reputable vendor	•synchronize updates on multiple servers	•synchronize service and updates	•use proven input source
Availability	•design and test to meet performance requirements	•plan and test for realistic system usage	•redundancy	•use redundant systems during maintenance	•backup power source
Safety	•control mods. •audit mods.	•check output before usage	•test procedures for clarity	•manual override for abnormal environment	•provide input checks

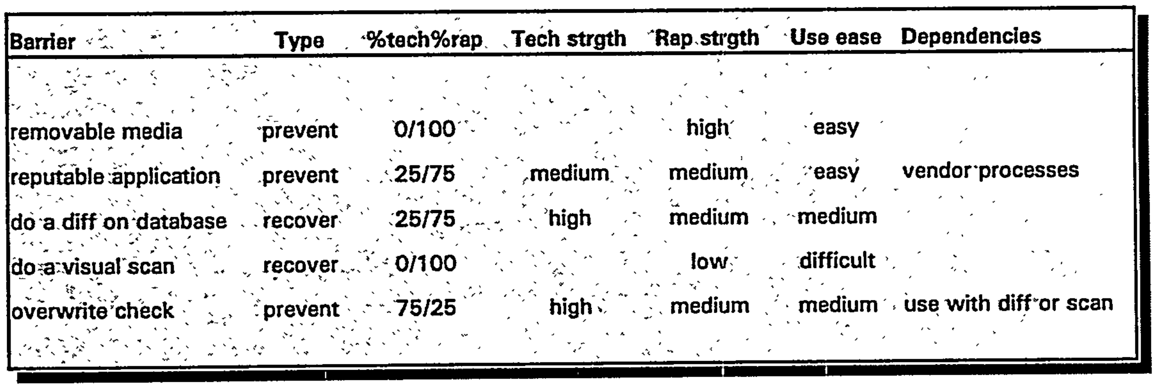
Figure 3. Risk Mitigators Matrix Example: Medical Decision Support System

In our example, the availability risk relative to system composition due to single points of failure could be mitigated by redundancy in important (or less reliable) system components (Figure 3). A different type of risk, an inappropriate process causing a utility risk, would be addressed very differently. A mitigator might be to use a thoroughly tested process obtained from a reputable vendor. The effectiveness of this type of mitigator depends partly on how well we understand what the process will be expected to do and how well it meets that need.

4.3 Barrier Analysis

Barrier analysis is the instantiation and refinement of a risk mitigator's ability to mitigate system specific risks. While the analyst may draw on the Risk Mitigators Matrix for nominal information on barriers, the information may need to be adjusted or supplemented to reflect how the barrier will function in the particular system at hand. Instantiation means the analyst should think about how well the barrier is likely to work given all that is known about the rest of the system, the way it will be installed, etc. For example, a very strong password generator becomes very weak if guest accounts are present, or if it is installed in such a way that it can be bypassed.

The analyst may assess the individual characteristics of risk mitigators: how much technology vs. how much rules-and-procedures (rap) are involved, perceived strength, cost to implement, ease of use, outside dependencies, etc. (Figure 4). One could use a scale of Low, Medium, or High for most of these. For hand analysis, these may all be considered informally, and a single figure-of-merit for the barrier may be used. This could be on a scale such as that suggested in section 4.5.5. The analyst must also identify in the model which risk state transitions are affected by each mitigator.



Barrier	Type	%tech%rap	Tech strgth	Rap strgth	Use ease	Dependencies
removable media	prevent	0/100		high	easy	
reputable application	prevent	25/75	medium	medium	easy	vendor processes
do a diff on database	recover	25/75	high	medium	medium	
do a visual scan	recover	0/100		low	difficult	
overwrite check	prevent	75/25	high	medium	medium	use with diff or scan

Figure 4. Barrier Analysis Spreadsheet

4.4 Threat Analysis

The project did not develop the threat agent concept. This concept is suggested for further development. Threat agents may be active or passive. Active threat agents may have characteristics such as motivation, skills, knowledge, time and other resources. They are willing to incur some amount of cost and risk, in order to gain the perceived value of their target. Passive threats, representing unintentional faults in the system, may have other characteristics. In either case, as a threat unfolds into the system, the agent's characteristics and the system's surety elements may both be altered by the interactions. When this concept is developed, then the individual characteristics of the mitigators can be made to interact with the individual characteristics of the threat agents. Threat agents can be run through the model as different threat scenarios.

In the absence of the threat agent definition, threat analysis just collapses to assigning global transition probabilities between risk states. Likewise, the barrier analysis may just as well be collapsed into a single figure-of-merit versus recording figures-of-merit for each of the individual characteristics.

4.5 System Risk Model

Before describing our method for building a system risk model, we explain the concept of influence diagrams.

4.5.1 Introduction to Influence Diagrams

An influence diagram is a probabilistic network that consists of nodes and arcs [Howard and Matheson, 1984; Jae and Apostolakis, 1992; and Jae *et al.*, 1993]. It must be singly connected and acyclic. The nodes can represent system states, decisions, or chance or deterministic occurrences, while the arcs represent the conditional dependencies between these occurrences. Decisions are represented by square nodes, while chance and deterministic events are represented by circular and double-circular nodes, respectively. These nodes ultimately influence a "value node" (diamond), which quantifies consequences for each possible system configuration (Figure 5).

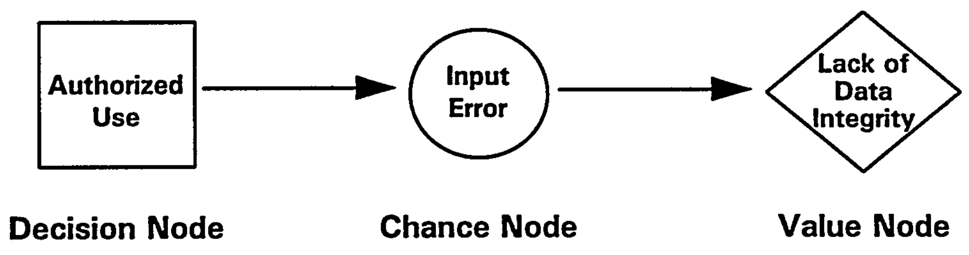


Figure 5. Basic Influence Diagram

If a chance node is dependent upon other nodes, it represents a *set* of conditional probabilities, where the probabilities are conditional upon the results of the node's immediate predecessors. A deterministic node is a special case of a chance node where all probabilities are either zero or one. Thus, an influence diagram consists of four distinct parts: the nodes, the influences upon the nodes (the dependencies between them), the "function" that determines which probabilities are to be applied given each distinct set of influences, and the conditional probabilities themselves.

The influence diagram formalism is conceptually similar to the event tree and decision tree formalisms that are applied in various disciplines of risk analysis. There are three distinct advantages to influence diagrams for information surety analyses. First, it is much easier for an analyst to visualize and gain an intuitive understanding of an influence diagram model than a comparable tree-based model. Second, influence diagrams show the dependencies between events explicitly, while this information is hidden from the analyst in event trees and decision trees. Finally, the influence diagram formalism supports both "backward" and "forward" construction of the model, while event trees and decision trees can only be constructed in the "forward" direction. The "backward" model construction process is similar to the method used for fault tree development in which an analyst uses deductive logic to systematically decompose a selected event to determine its root causes. The "forward" model construction process is used in event tree analysis to inductively determine the logical consequences of a particular event or set of events. The influence diagram formalism supports both model construction methods. This flexibility and power provides an important reason to use influence diagrams instead of either fault tree or event tree/decision tree methods.

Conventional influence diagrams have three primary disadvantages when applied to information systems. First, the node symbology, because it is so general, does not reflect some of the ideas that we believe are important to represent in a risk model of an information system. Second, the traditional solution method does not show or even generate the detailed set of scenarios or paths possible in the model [Jae and Apostolakis, 1992; Shachter, 1986]. The ability to examine these paths in detail is a primary advantage of the event tree and decision tree methods. Finally, the traditional solution method makes it difficult to determine which nodes are the most important for various aspects of the results and, hence, to determine where one should look to improve the system. This very useful information is key to the results of typical fault tree analyses. We have therefore proposed some extensions to the conventional influence diagram notation and a new solution methodology to remove these deficiencies.

4.5.2 Building a Risk Model of an Information System

Our objective in modeling an information system is not so much to provide a "probability of failure" for a system as it is to help identify and prioritize that system's risks. Only after the risks have been identified and prioritized can an analyst make informed decisions about whether particular risks are acceptable and, if necessary, examine strategies to reduce those risks.

The starting point for a risk model should be to identify the consequences and benefits from the proper or improper operation of the information system. One should consider each "information surety objective" (safety, functionality, availability, confidentiality, and integrity) as it might affect risk areas such as the mission of the system or organization; worker health and safety; public health and safety; as well as legal, regulatory, political, social, and environmental impacts. This assessment helps identify which system states should be either encouraged or avoided, and forms the basis for the risk assessment model.

There are many ways to build a risk model based upon an influence diagram, but two of the most useful are as follows. Under the first method, we explicitly identify the undesired state and work to find the immediate, necessary, and sufficient conditions for that state to occur in much the same manner as during fault tree construction. We continue to apply these criteria recursively until each event has been resolved into its fundamental causes along with the system conditions required for these causes to successfully act upon the system. These fundamental causes (basic failures and initiating events) will form the starting point in our search for ways to reduce or eliminate particular system risks.

The second method for developing influence diagrams begins by drawing a diagram to represent the normal functional flow of the system (including the hardware, software, and data aspects of the system). We then examine every node to find influences that can cause the system to deviate from normal functionality toward an undesired state. We add events to the influence diagram to represent these influences, and seek to find their fundamental causes in much the same way as would occur for the first method. In addition, if we suspect *a priori* that particular events or conditions might lead to an undesired state, we can use these nodes as starting points (initiating events) and expand them forward into their universe of logical consequences to determine how they influence the normal and even the abnormal operation of the system. This shows the value of working both "forward" and "backward" when developing influence diagram models.

Another powerful feature of influence diagrams is that, like fault trees and other modular directed graph techniques, they can support iterative refinement. Thus, it is possible to initially construct a "high level" model for scoping studies using only a few broadly defined nodes, and to later refine the model to incorporate a more detailed knowledge of the system, its operation, and its vulnerabilities. It is also possible to construct a model in which some phenomena are examined only in coarse detail (a "screening analysis") and others at a much finer level.

So far, our risk model only considers influences that can lead us toward undesired states. We must also consider "positive" influences that can reduce the ability of "bad" influences to accomplish an undesired result. These are called "barriers" because they act as impediments to undesired outcomes much like a fence system acts as a barrier to prevent unauthorized access to a facility. Barriers show up as influences (nodes with appropriate arcs) in the influence diagram. Since a barrier node typically depicts whether a particular barrier is present or active, it is often represented as a chance node that is not influenced by any other nodes (i.e., an unconditional chance node), where the probability value represents the likelihood that the barrier is *implemented*, and *not* the probability that it is *effective*. Barrier effectiveness is modeled in the node that the barrier influences, and takes the form of the conditional probability that a particular influence will actually cause the system to deviate from its intended function. This allows us to assess the effects of multiple barriers on a single "bad" influence both individually and in combination. For example, we could consider controlling access to a facility using badges, passwords, and biometrics (e.g., hand measurements, retina scans). Our node that represents the chance of a person being granted inappropriate access would be influenced by four nodes that represent: (1) a person trying to gain access (if nobody wants access, we don't have any risk and don't need any protection), (2) the presence or absence of a badging system, (3) the presence or absence of a password system, and (4) the presence or absence of a biometric access control system. The effectiveness of each combination of barriers is measured by the conditional probability that a person is granted inappropriate access. This probability varies depending upon which combination of controls (barriers) is in use.

Initiating events, basic failures, and barriers (engineered or otherwise) are not conceptually different from other chance or deterministic events. However, using different influence diagram symbols to represent these events would help the analyst to more quickly assess the completeness of the list of threats (initiating events and basic failures) and to identify any unmitigated threats (paths without barriers to undesired consequences). Therefore, we propose to extend the traditional influence diagram symbol set as shown in Figure 6. We propose using a house symbol to represent initiating events (as these can be likened to external events in a fault tree analysis, whose symbol they would share), and a triangle to represent a barrier. A basic failure would be represented by a circle, as it is essentially a chance occurrence (and is similar to a basic event in a fault tree analysis, which is also represented by a circle). It is possible to build a risk model of an information system without these additional symbols. However, we believe that their use will give additional scrutability and utility to the risk models.

If the influence diagram is to be solved mathematically, we must develop true logical interrelationships between the nodes as well as quantitative probabilities for those relationships. The logical relationships can be thought of as an "If — Then — Else If — Else" block from a computer programming language. The conditions for the "If" and "Else If" blocks are used to express exactly how this node is related to the nodes that it is influenced by. The content of the "Then" and "Else" blocks is the probability information for the current node <u>*given that*</u> (conditional upon) the specific logical conditions being satisfied. Visually it is easier for many to enter and understand these logical relationships in the form of a truth table. This would be our preferred implementation of the logic definition input software.

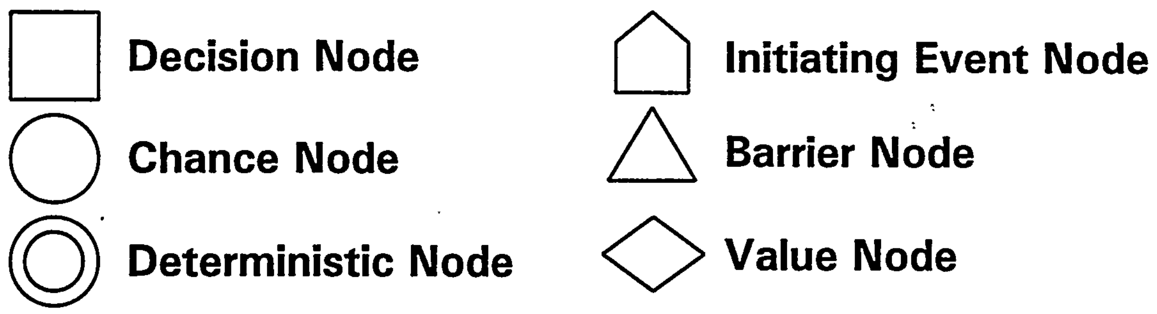


Figure 6. Extended Influence Diagram Symbols

4.5.3 Constructing a Sample Risk Model

Consider an application in which a robotic system is to automatically lift and move a large, heavy object. The system is composed of a crane-type hoist system and an automated controller (an "information system"). We assume that there are humans nearby who can intervene in the process should it go awry. However, if we expect them to intervene, we must develop and implement appropriate procedural instructions.

Our risk assessment begins by identifying the consequences and benefits from the proper or improper operation of the system. The system designer wants to use an information system to control the lift activity to reduce the time required to stabilize a swinging load before it can be set down. Potential consequences include lost productivity (setting the load in the wrong place so that it must be moved again), worker injury (the load hits or is set down upon a worker), system unavailability (the system must be operable when it is needed to prevent lost productivity), and, if the system is to move hazardous loads or operate near hazardous areas, public safety and environmental issues. While it is beyond the scope of this paper to develop each of these potential consequences, it is apparent that the more serious consequences are caused by the system moving, setting, or spilling something into an undesired location. This consequence will be used as the basis for the "value" node at the endpoint of our sample influence diagram.

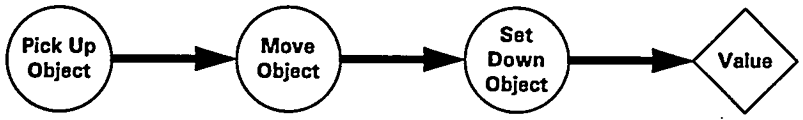
We continue to build the sample risk model by constructing a model of the system's "normal flow of operation" and looking for influences that can cause deviations from that normal flow. Figure 7 shows this process in three steps. For this simple example, we model the system's normal operational procedure as a lift operation, a move operation, and a set-down operation. These operations must occur in sequence, so each operation's node on the diagram is influenced by its predecessor in the normal flow of operation.

The next step is to identify the influences that can cause the system to deviate from the normal flow of operation toward the consequences that we have identified. Step two in Figure 7 shows a number of influences that can cause this to occur, and the aspects of the normal flow of operation that they would affect. Owing to space considerations, this list is by no means complete. Note, however, that the "controller fault" node is not sufficiently detailed for assessment. Thus, we refine this node to incorporate its underlying causes just as we would successively refine the failures in a fault tree. Note also that both "initiating events" (e.g., bad software) and "unconditional random failures" (e.g., controller hardware failure) can influence the system away from its intended mission.

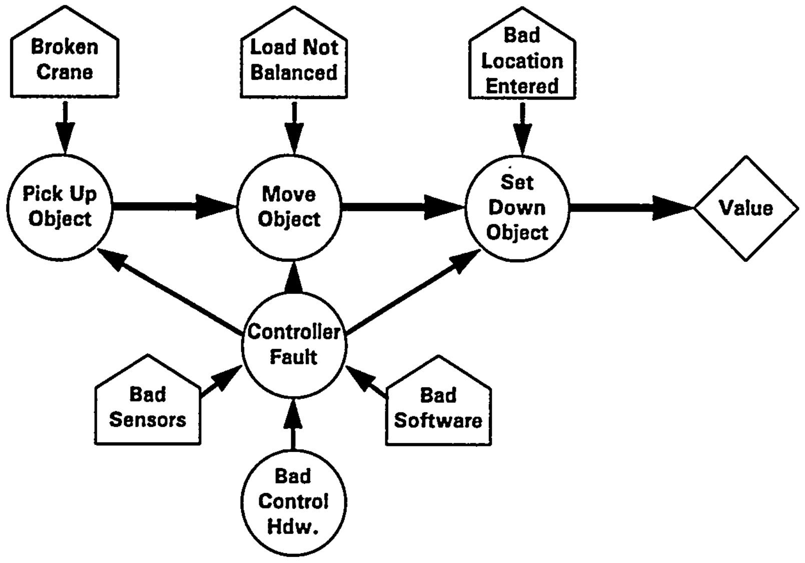
The third step is to incorporate "barriers" into the model. These influences act to reduce the probability that a "bad" influence will produce an undesired result. Step three in Figure 7 shows a number of potential barriers that might reduce system risk. Some barriers are procedural (e.g., preoperation checklists and visual verification of the ongoing process), while others are technological (e.g., a fault-tolerant controller hardware and an automatic position verification tool). The list of barriers should be viewed as providing *design options* that can be implemented individually or in combination.

This risk model provides some general modeling insights. First, a single barrier may affect more than one operation, initiating event, or threat. For example, a preoperation checklist might act both to identify damaged equipment and to ensure that the load is balanced. This is easily represented in the model. It is also possible to use more than one barrier against a single threat. For example, both an automatic position verification tool and visual monitoring of the process can act as barriers to ensure that an improperly entered target location does not cause the load to be taken to the wrong location. Finally, note that this model still contains one "unmitigated threat" in that the initiating event "bad sensors" is not directly mitigated by any barrier in the model (although, depending upon the design, the visual monitoring of the process might help to mitigate the *effect* of a sensor failure). Upon recognizing an unmitigated threat, a designer can either knowingly accept the risk posed by that threat or find a barrier to effectively mitigate it. This is, however, a conscious decision based on an identified risk, rather than a default design based upon ignorance (as might occur without the information provided by an analysis such as the one proposed here).

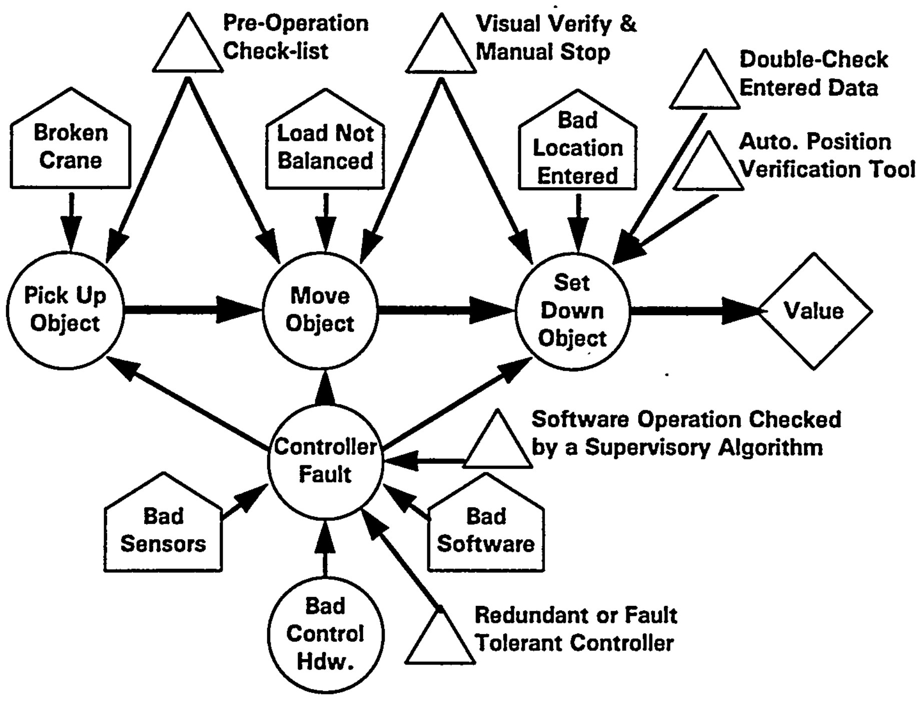
One of our objectives in designing a risk modeling tool is that it be readily extendible. This sample risk model seems to fit this criterion. The level of detail in this sample model might be appropriate for a high-level scoping study. If a more detailed study were required later, one could, for example, break down the "Controller Fault" node into a greater level of detail. This might identify new fault conditions and initiating events, and provide an opportunity to include new barriers as the system is better understood. We believe that models of this type can be extended to an arbitrary level of detail in much the same way as one would extend a fault tree analysis. This property makes this method a valuable tool for performing many types of risk assessment studies.



Step 1. Establish normal functional flow of the system.



Step 2. Find influences that cause the system to deviate from normal functionality.



Pre-Operation Check-list
Visual Verify & Manual Stop
Double-Check Entered Data
Auto. Position Verification Tool
Broken Crane
Load Not Balanced
Bad Location Entered
Pick Up Object
Move Object
Set Down Object
Value
Controller Fault
Software Operation Checked by a Supervisory Algorithm
Bad Sensors
Bad Software
Bad Control Hdw.
Redundant or Fault Tolerant Controller

Step 3. Find barriers to mitigate undesired conditions.

Figure 7. Example Risk Model Construction

4.5.4 Risk Matrices and the System Model

The example system model developed for the robotic system made use of a model construction technique in which we look for deviations from the normal flow of system operation. This is but one of several methods that can be used to construct the influence diagram system model. However, the methods we have described so far do not have an obvious connection to the Risk Identification Matrix and the Risk Mitigators Matrix that were described in the previous chapter. It is important that there be a consistency between the matrices and the system model because each represents a different way of looking at the information surety problem and may identify different surety issues. This section outlines how the risk identification and mitigation matrices can be used in the system model construction process, and how the model results can be used to make additions and refinements to the matrices in an iterative process.

Iterative refinement is basic to the way in which an analysis should be carried out. The analyst should first work in the context of a high level system risk model, input estimates, run the analysis engine, and examine the results. High risk paths can then be strengthened with additional barriers, or can be broken down into more detail. Highly uncertain paths that are determined to be of sufficiently high consequence call for better estimates, and they may also benefit from refinement. Once the highest level risks have been adequately addressed, the analyst may wish to incorporate additional risks into the model and continue the analysis. The analyst will be able to see the total impact of old and new barriers on all risks.

The process of performing an information system surety analysis often begins with an informal assessment of surety requirements and potential problems. As the requirements are formalized, one could begin to fill out the Risk Identification Matrix and treat it as a formalized list of potential information surety issues to which the system is to be exposed. Going through the established process of completing the risk identification and mitigation matrices provides a preliminary "risk analysis" of the system in much the same way that a "Failure Modes, Effects and Criticality Analysis" (FMECA) provides preliminary risk information for certain mechanical systems.

The next step in the analysis is to construct a system model using the influence diagram technique. The matrices can then be examined to determine whether each of the identified sources of risk and barriers is found in the system model (either as an individual node or as a combination of nodes) and, if so, whether the interconnections between the nodes accurately represent both the real system and the issues identified in the risk matrices. The risk matrices can also be updated based on any new risk sources or mitigators that were identified during the construction of the system model. Both of these updating process can spawn new ideas — either for the matrices, the system model, or both. Thus, this can be an iterative process. When used in this way, the matrices serve as an essentially independent completeness check for the system model, as does the system model for the matrices. Automated tools can easily keep the two in sync. This can be especially useful when constructing the system model based on the "deviations from the nominal flow of operation" method. It assures that risks and mitigators that may not obviously cause the system to deviate from nominal functionality are still represented in the system model (matrix view) while still forcing the analyst to make a detailed consideration of all phases of nominal operation (model view).

There is a second method for constructing a system model in which the matrix information is used directly during model construction. This method was described earlier as looking for the immediate, necessary and sufficient conditions for each undesired condition of the system (without specifically modeling the normal flow of system operation). Under this method, each risk identified in the Risk Identification Matrix is either a cause of an undesired condition or is itself an undesired condition. Thus, we begin construction of the system model by finding all of the undesired conditions that are either found in or implied by the Risk Identification Matrix. Each of these conditions is placed in the system model as a node immediately preceding the value node as it is a direct contributor to our view of the system's worth as expressed in the value node. We then examine each node and seek to determine the immediate, necessary and sufficient conditions for its occurrence and either connect appropriate existing nodes or create new nodes as necessary to represent those conditions (influences on whether the state represented by our node actually occurs). Many of the new nodes that are to be created will either be found directly from entries in the matrices or be implied by them, but it is also important at each stage to consider how issues not found in the matrices can influence the system (these may be factored back into the matrices as appropriate). Finally, it is important to consider known and proposed barriers at each step that can help prevent the realization of the undesired state. Again, many of these barriers will come from the risk mitigation matrix, but we must also consider influences that have not been identified in the matrix. We continue to apply these steps recursively until each node has been resolved into its fundamental causes along with the system conditions required for these causes to successfully act upon the system.

4.5.5 Quantification of Node Probabilities

We have to this point minimized an important facet of the quantitative solution of the system models: how do we generate real and believable probability values for the nodes and conditions that we will find in our system models? If the system being analyzed already exists, it may be possible to collect real operational data for use in the model. If the system is still under design, it may be possible to collect data for some items from comparable existing systems or from early development prototypes. It is also possible to obtain probabilistic information for some types of model nodes by breaking down the state that the node represents into a series of more basic probabilistic questions that can be answered directly, with the node probability being derived by aggregating the simple results. As with all measured and modeled data, the probability values generated by any of these processes are subject to statistical validity limitations and are based on an imperfect state of knowledge about the real system. For these reasons it is very important that the model be quantified in the light of a detailed uncertainty analysis.

Frequently a model will contain nodes for which there is no existing data or for which it is impossible or impractical to collect data. In these cases, the probabilities must be estimated by experts in the relevant field. This process is known in the literature as "expert judgment elicitation" and can be as informal as an analyst providing detailed documentation for what reference materials and models were considered when a particular probability value was selected. It can also be a highly formalized process in which subject experts are contracted from the outside, provided with statistical training (intended to "debias" the experts), and asked to provide formally documented individual results which are then aggregated across a spectrum of experts and review groups [USNRC, 1990]. It is especially important for an uncertainty analysis to be performed when probability values are based upon expert judgment because the uncertainty analysis results can be used to indicate how sensitive the results are to the particular value chosen by the experts.

One technique for assisting in the expert judgment process is the method of probability intervals. Probability intervals can be used to represent our knowledge and uncertainty about the performance of individual surety components. The performance metric to be used is the probability of effectiveness of the component performing its intended function in the operational environment. For example, when security is the surety dimension being evaluated, the following probability evaluation scheme for safeguards effectiveness has been successfully used in various applications. The zero to one probability interval is divided into five regions, and each region is represented by a categorical probability value:

0.10 - adversary is heavily favored to win
0.25 - adversary is favored to win
0.50 - neutral or competitive situation where either
the adversary or safeguards could win
0.75 - safeguards is favored to win
0.90 - safeguards is heavily favored to win

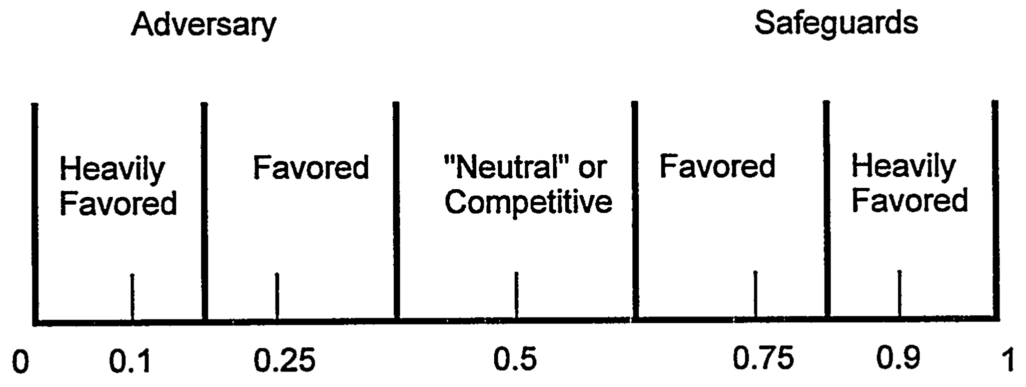


Figure 8. Probability Interval for Safeguards Effectiveness Evaluation

These categorical probability values can be thought of as comprising the region from which experts might choose a number to assign after they determined which of the five situations best characterized a particular event. (The boundaries of each region are those points that are equidistant between each categorical value and its neighbors.) This degree of numerical differentiation is the kind that can be obtained practically when using expert judgment probability assessment techniques.

Hence the probabilities of safeguards component effectiveness can be estimated using these categorical probability values. Let $p_i(e)$ be the probability of effectiveness for safeguards component i. Then $p_i(e)$ takes on a categorical probability value of 0.10, 0.25, 0.50, 0.75, or 0.90 depending on the corresponding scenario context. Each probability estimate corresponds to a well-defined event including the following key details:

- Relevant access and authority characteristics of adversary
- Particular defeat method being employed
- Relevant characteristics of the safeguards guarding against the defeat method

The overall probability of system success in protecting against each scenario is based on the probabilities of effectiveness of the combination of safeguards components invoked by the scenario. Let $P_x(SWin)$ be the probability of the safeguards system winning against scenario x. Then,

$$P_x(SWin) = 1 - \prod_{i \varepsilon J_x} [1-p_i(e)] \quad \text{where}$$

$p_i(e)$ is the probability of effectiveness of safeguards component i
J_x is the set of safeguards components protecting against scenario x

Since $P_x(SWin)$ is a function of categorical probabilities, its evaluated value can be interpreted using the same scale. If desired, some function of the $P_x(SWin)$s may be derived to yield a measure of total system risk.

The notion of categorical probabilities is particularly suited for representing context-rich environments when there is sparse statistical data, as in safeguards evaluations and safety assessments which are scenario-based.

4.5.6 Enhanced Methods for Solving Influence Diagrams

Many past applications of influence diagrams have been in the area of decision theory, where an analyst wants to determine which decision option will lead to the greatest possible utility or the lowest possible consequences. Thus, influence diagrams have been solved by successive simplification of the network using arc reversal and node removal based on probabilistic rules [Jae and Apostolakis, 1992; Shachter, 1986]. The objective is to obtain a network that consists of only two nodes (one representing the decision and one representing the "value" -- utility or consequences). Under such a solution method, the probabilistic and deterministic information is systematically combined until all that remains is the conditional probability that each possible decision will result in each of the possible outcomes. While this works well for some decision analyses, it provides little guidance, for example, as to where one might be able to improve the system given additional resources.

We believe that a risk assessment should provide not only a quantitative estimate of risk (the reliability of the actual quantitative value is always questioned), but also information such as a list of the system's most likely and riskiest paths; a ranking of the events, nodes, probabilities, and uncertainties that figure most prominently in the risk of the system; and a list of places where improvements in the system will lead to the greatest risk reduction. Our objective is to develop a solution method that allows us to obtain this information from an influence diagram. Similar information is routinely generated in event tree analyses (the list of event tree paths) and in fault tree analyses (the cut set importance measures). Therefore, we have explored the adaptation of fault tree and event tree solution methods to influence diagrams.

Other studies [Jae and Apostolakis, 1992] have demonstrated that it is always possible to translate an influence diagram into an event tree or a decision tree (this is not a one-to-one translation, as there are often many appropriate event tree representations for a given influence diagram). The conversion from influence diagram to event tree can be done using the following process:

1. Make a list of all nodes in the influence diagram. The list may be in arbitrary order, but must include the final "value node(s)." This list will contain the nodes remaining to be implemented in the event tree (the "remaining" list). Make a second empty list. This list will contain the nodes that have already been implemented in the event tree (the "implemented" list).

2. Move any value node(s) from the "remaining" list to the "implemented" list. If there is more than one value node, they may be moved in arbitrary order. These nodes will be the first to appear on the "implemented" list, and will eventually be the last events in the event tree model (we are building the event tree from the end to the beginning).

3. Make a pass through the "remaining" list. For each entry in the list, examine the entry to determine whether all of the nodes that are influenced by this node ("downstream nodes") are already on the "implemented" list. If this is true, then this node may be moved to the "implemented" list and placed at the bottom of that list.

4. Make repeated passes through the "remaining" list per the method of Step 3 until all nodes have been moved to the "implemented" list (the "remaining" list is empty). At this time, the "implemented" list represents the reverse of the order in which the nodes will be evaluated in the event tree model.

5. Write the event tree logical input for the last entry on the "implemented" list to the event tree input file. Remove this entry from the list.

6. Repeat Step 5 until the "implemented" list is empty. For each entry (influence diagram node), we must write the logic that connects this node (event in the event tree) to the nodes (prior events) that it is influenced by. When the "implemented" list is empty, event tree representation of the influence diagram is complete.

The typical event tree solution process involves an exhaustive consideration of all possible paths, with the removal of physically or logically precluded paths. Solving an influence diagram in this manner is conceptually very simple and generates important results that cannot be obtained using traditional influence diagram solution methods (such as successive simplification).

The event tree solution method, while an improvement over traditional solution methods, does not generate all of the types of event importance information that we would like to obtain from our risk analysis. We would like to obtain event importance information for influence diagrams and event trees that are similar to those generated during fault tree analyses. We have been prevented from doing so because the fault tree importance measures are all based upon the assumption of binary Boolean variables while event trees and influence diagrams often contain multi-state logic (i.e., in a fault tree, an event is either true or false, while in an event tree or influence diagram, the event can take on multiple discrete values). At Sandia National Laboratories, we have discussed mathematical methods for developing multi-state event importance measures [A.C. Payne, Jr., personal communication]. The method, which has been demonstrated on simple sample problems, can be summarized for binary events as follows and will be later described for multi-state events:

First, consider each event tree path to be equivalent to a cut set from a fault tree analysis. Each event that occurs in that path (each probability that contributes to the path) can be thought of as a basic event in the cut set. If each event tree question is limited to two possible outcomes, the group of cut sets that represent the event tree paths can legitimately be examined using all traditional cut set importance measures such as the partial derivative, risk increase, risk reduction, and Fussell-Vesely importance measures [Roberts *et al.*, 1981]. Traditional cut set uncertainty analysis techniques (e.g., Monte Carlo and Latin hypercube sampling techniques) can also be applied without adaptation [Iman and Shortencarier, 1984, 1986]. Therefore, we can extract all of the information we need from an influence diagram if we solve it as an event tree, translate the paths into cut sets, and evaluate them using traditional fault tree cut set importance measures.

The only restriction on this technique is that any event tree that is developed from the influence diagram must be constructed using only two outcomes per question (binary events). This does not represent a *theoretical* limitation on the methodology because other studies have demonstrated that there exists at least one "binary event tree" for each "multibranch event tree." It is, however, a *practical* limitation on the method because we may want to let a single node in the influence diagram assume multiple states. For example, the "Move Object" node in our sample problem might take on the values

"Normal Move," "Too High," "Too Low," "Too Fast," and "Quivering." We would like to be able to translate this node into a single event tree question with multiple outcomes, but are prevented from doing so if we want to obtain event importance information. The translation from multibranch events to binary events is difficult and results in an event tree that is far more difficult to understand than the original multibranch tree.

Sandia has conducted some research directed toward developing importance measures for "cut set" expressions involving nonbinary events. This currently unpublished research indicates that importance measures similar to the partial derivative, risk increase, and Fussell-Vesely importance measures can be calculated with only slight modifications in the computational algorithm from their binary event counterparts. If certain minor additional computational assumptions are made, a measure that parallels the risk reduction importance measure can also be calculated for non-binary events. However, the uncertainty and uncertainty importance analyses for a cut set expression for multiple-outcome events are complicated by the fact that the probabilities for all outcomes for a particular event, when summed, must equal one. It has been demonstrated elsewhere that this problem can be solved using the multivariate Dirichlet distribution [Payne and Wyss, 1994].

4.5.7 Conceptual Example Computation

The influence diagram shown as Step 3 in Figure 7 contains a value node, five chance nodes, five initiating event nodes, and six barrier nodes. If we apply the method from the preceding section to convert this influence diagram, we might proceed as shown in Table 1. As each node in the influence diagram becomes an event in the event tree representation, the event tree for this diagram will have a total of 17 events (questions) -- one for each node in the diagram.

Table 1 (part one of four). Conceptual Example Computation

Step 1 Lists - "Remaining" list in arbitrary order with an empty "implemented" list.				
Node No.	Remaining	Influ-ences	Infl. By	Implemented
1	Pre-Operation Check-List	8,9	-	
2	Visual Verify, Man. Stop	9,10	-	
3	Double-Check Data	10	-	
4	Position Verify Tool	10	-	
5	Broken Crane	8	-	
6	Load Not Balanced	9	-	
7	Bad Location Entered	10	-	
8	Pick Up Object	9	1,5,12	
9	Move Object	10	1,2,6, 8,12	
10	Set Down Object	11	2,3,4, 7,9,12	
11	Value	-	10	
12	Controller Fault	8,9, 10	13,14,15 16,17	
13	Supervisory Algorithm	12	-	
14	Bad Sensors	12	-	
15	Bad Control Hardware	12	-	
16	Fault Tolerant Controller	12	-	
17	Bad Software	12	-	

Table 1 (part two of four). Conceptual Example Computation

Steps 2, 3 & 4 Lists - Partially completed moving entries to the "implemented" list.

Note: This snap shot is taken part way through the third pass through the "remaining" list, immediately before considering Node 8. Our next step is to move Node 8 to the "implemented" list. This will allow us to move Node 12, which will in turn allow us to move Nodes 13 through 17 to the "implemented" list. Thus, after the third pass through the list, only Node 1 and Node 5 will be on the "remaining" list.

Node No.	Remaining	Influ-ences	Infl. By	Implemented
1	Pre-Operation Check-List	8,9	-	11. Value
2	*Moved during Pass 3*			10. Set Down Object
3	*Moved during Pass 2*			3. Double-Check Data
4	*Moved during Pass 2*			4. Position Verify Tool
5	Broken Crane	8	-	7. Bad Location Entered
6	*Moved during Pass 3*			9. Move Object
7	*Moved during Pass 2*			2. Visual Verify, Man. Stop
8	Pick Up Object	9	1,5,12	6. Load Not Balanced
9	*Moved during Pass 2*			
10	*Moved during Pass 1*			
11	*Value Node - Moved Initially*			
12	Controller Fault	8,9, 10	13,14,15 16,17	
13	Supervisory Algorithm	12	-	
14	Bad Sensors	12	-	
15	Bad Control Hardware	12	-	
16	Fault Tolerant Controller	12	-	
17	Bad Software	12	-	

Table 1 (part three of four). Conceptual Example Computation

Step 4 Lists - All entries moved to the "implemented" list.				
Note: It took 4 passes through the list to completely empty the "remaining" list. A different tree may require more or less passes through the list to accomplish this because of how the nodes are interconnected. The "implemented" list, when taken in reverse order, now represents an acceptable order for the events in the event tree representation of the influence diagram. Recall that the events can occur in any order as long as each event comes before any other event that it influences.				
Node No.	Remaining	Influ-ences	Infl. By	Implemented
1	*Moved during Pass 4*			11. Value
2	*Moved during Pass 3*			10. Set Down Object
3	*Moved during Pass 2*			3. Double-Check Data
4	*Moved during Pass 2*			4. Position Verify Tool
5	*Moved during Pass 4*			7. Bad Location Entered
6	*Moved during Pass 3*			9. Move Object
7	*Moved during Pass 2*			2. Visual Verify, Man. Stop
8	*Moved during Pass 3*			6. Load Not Balanced
9	*Moved during Pass 2*			8. Pick Up Object
10	*Moved during Pass 1*			12. Controller Fault
11	*Value Node - Moved Initially*			13. Supervisory Algorithm
12	*Moved during Pass 3*			14. Bad Sensors
13	*Moved during Pass 3*			15. Bad Control Hardware
14	*Moved during Pass 3*			16. Fault Tolerant Controller
15	*Moved during Pass 3*			17. Bad Software
16	*Moved during Pass 3*			1. Pre-Operation Check-List
17	*Moved during Pass 3*			5. Broken Crane

Table 1 (part four of four). Conceptual Example Computation

Possible Event Tree Questions for the Example Problem	
Note: This table is provided as an aid to those who are already familiar with event tree modeling. It shows the event tree questions that might be asked to correspond to the nodes in the influence diagram. If the conversion were performed automatically by software, the questions would likely only contain the node names from the influence diagram or any supporting comments provided by the analyst.	
#	Question
Q1	Is the crane broken before the move is started?
Q2	Is there a pre-operation check-list found any problems with the crane or with the proposed move?
Q3	Is there defective software in the controller?
Q4	Is the controller fault-tolerant to compensate for possible control hardware failures?
Q5	Is the control hardware in a state of failure?
Q6	Are all of the necessary sensors functioning?
Q7	Is there a supervisory algorithm for the controller that can shut down the proposed move if it sees an unacceptable condition?
Q8	Considering Q3 through Q7, does the controller function successfully?
Q9	Considering Q1, Q2 and Q8, is the object successfully picked up?
Q10	Is the load balanced?
Q11	Is there an effective procedure to visually verify the load balance and other critical move and set-down parameters, and stop the move if anomalies are found?
Q12	Considering Q2, Q8, Q9, Q10, and Q11, is the movement of the load successful?
Q13	Was a bad final location entered into the controller?
Q14	Is there an automatic tool available to verify that the final set-down position is in fact a legal location for this load to be set down?
Q15	Is there a procedure in place to have the operator manually double-check that the entered final desired location does not contain errant data?
Q16	Considering Q8, Q11, Q12, Q13, Q14 and Q14, is the moved object successfully set down in the right place?
Q17	Considering Q16, what are the costs and benefits associated with this move?

If each node in this diagram were to be considered a binary event, the theoretical maximum number of paths through the event tree would be 2^{17} (or 131,072 paths). Using multiple-outcome events would cause this number to increase rapidly. While the theoretical maximum number of paths through an event tree may be large, the actual number of paths realized in the event tree solution may be much smaller. The actual number of paths generated will depend upon such variables as the number of possible outcomes for each node and whether some paths are precluded based on physical arguments. For example, if the crane is broken and cannot pick up the object, it is not possible for the crane to be carrying an unbalanced load. In addition, most event tree analysis tools allow the user to define a truncation probability so that paths of negligible probability can be eliminated. This allows the analyst to concentrate on inferring results from a number of paths that, while still quite large, is at least manageable.

It is clear that no analyst would want to graph an event tree with more than a couple of hundred possible end states — even with the assistance of an event tree graphics software tool [Camp and Abeyta, 1991]. Sandia's SETAC event tree analysis code suite [Wyss and Daniel, 1994], which is based on the EVNTRE code [Griesmeyer and Smith, 1989] provides an automated nongraphical facility for efficiently processing these large event trees. There is not yet a facility for converting the paths obtained by SETAC into cut sets for the performance of the cut set importance analysis described earlier. However, upon the development of such software, the computation of risk analysis results from a completed influence diagram would be a simple batch process that would take the following steps:

1. Convert the influence diagram and its associated logic and quantification information into an event tree representation as described previously.

2. For an uncertainty analysis, perform stratified Monte Carlo sampling of uncertain values using the LHS Latin Hypercube Sampling software.

3. For each sample observation generated by the LHS software, solve the event tree recursively using the SETAC event tree analysis software. SETAC interfaces directly with LHS, so a single SETAC run will solve the event tree for all sample observations. During the solution process SETAC must save all relevant path information so that the event and cut set importance analysis can be performed.

4. Use the path information generated by SETAC as cut sets as the input for a multi-state importance analysis as described previously using a variant of the TEMAC importance analysis software.

5. Examine the important cut sets and events to determine where new barriers or other design features may be required.

If necessary, the analysis process can be repeated in an iterative fashion with new barriers and risk sources being used to update the risk matrices and to produce a revised system model for further analysis. Especially in a design-phase analysis, this process should be repeated until all of the remaining risks are both understood and knowingly accepted by those who will be responsible for the system and its proper operation.

4.5.8 Summary

In this chapter we have seen how commonly used probabilistic risk analysis techniques can be used to provide insights into information surety issues. We discussed how an adaptation of the influence diagram formalism can be used as a more natural modeling tool for these types of studies. The methods described in this chapter are generally robust, and many have already been implemented in software. Finally, we demonstrated the major aspects of the methodology on a simple example problem.

5 SOFTWARE TOOL SUPPORT

The development of the software to support the methodology focused largely on how to obtain all the pertinent information from the user via the user interface. The concept was that data entered with the interface would be used to generate risk scenarios, and existing analysis codes could be used to quantify the scenarios. The purpose of the interface was to facilitate the collection of relevant data and to motivate the user to think about the problem from multiple dimensions. Our investigations have shown the validity of this assumption.

The interface was developed using principles that are often specified for the design of good graphical user interfaces. These principles express how the software does what it is supposed to do.

- ease of use
- graceful error recovery
- context-sensitive help at the user's request
- accurate reflection of the methodology
- prevention of incorrect data entry
- intuitively obvious use (as far as possible)
- use of commercial off the shelf (COTS) software for the various components

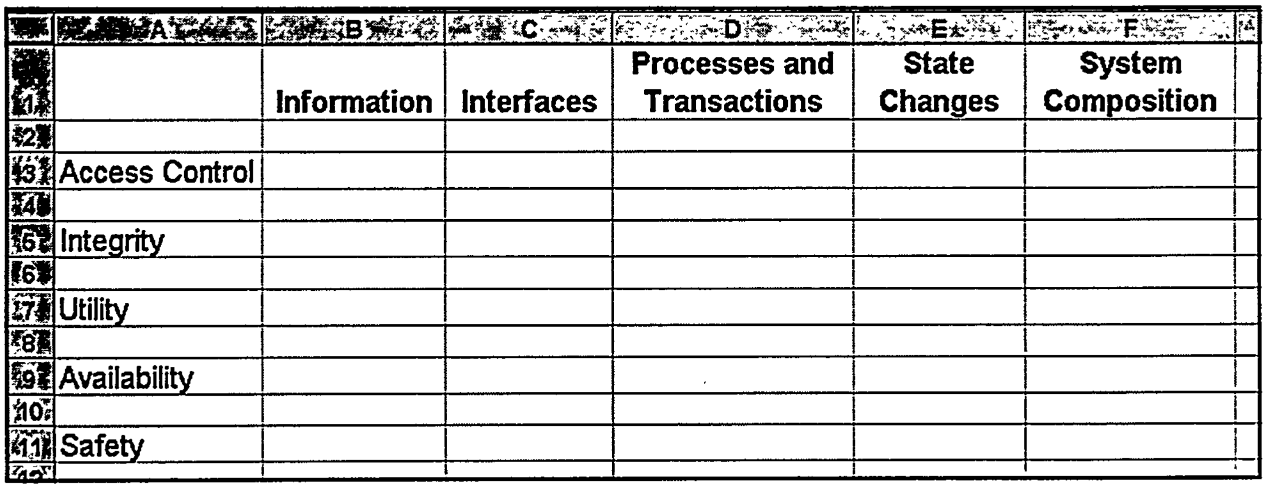
The specifics of what our risk assessment software should do include the following:

- provide the user with a generic risk taxonomy
- allow the user to create risk taxonomies unique to the information system based on the generic taxonomy, the user's input, or both
- modify risk taxonomies
- archive modified taxonomies
- allow the user to specify the risks particular to an arbitrary information system
- create a system model (the influence diagram) for the information system from the specified risks with as little additional input from the user as possible; this activity will be initiated via a menu pick or a button click and will take the form of a set of leading questions to ascertain the desired information
- highlight scenarios or paths of interest in the influence diagram
- highlight risk sources or mitigators of interest in the matrices

- analyze the information system to find weaknesses, unmitigated risks, unacceptable residual risk, areas with significant uncertainties, etc.
- indicate conflicting mitigators (i.e., one that decreases the risk in one area but increases the risk in another)
- save the state of the various risks, mitigators, consequence probabilities, consequence severities, and scenarios so that the user can vary parameters to gain additional insights (what-iffing)
- provide a tutorial to explain briefly the unique features of the methodology, how to use the software, unique vocabulary, etc.

Time constraints did not allow the development of all of these capabilities to their fullest potential. The following paragraphs describe those capabilities that were developed and how they function in the tool in its state at the end of our research.

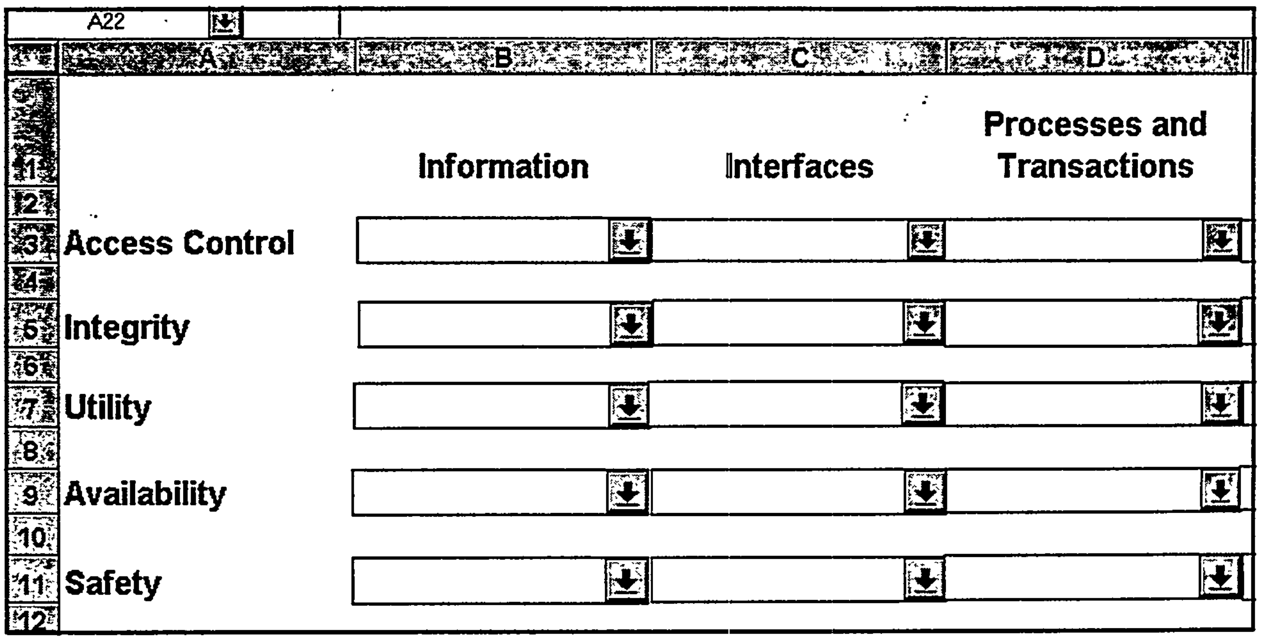
As discussed in previous sections of this report, the methodology involves identification of the information system risk sources, specification of risk mitigators, creation of the influence diagram to determine the interactions between the system and the sources of risk, analysis of the system surety risks, and modification to enhance information system surety. The goal of risk specification is to capture the significant risks to the system. The risk matrix contains these risks; an empty risk matrix is shown in Figure 9.



	A	B	C	D	E	F
1		Information	Interfaces	Processes and Transactions	State Changes	System Composition
2						
3	Access Control					
4						
5	Integrity					
6						
7	Utility					
8						
9	Availability					
10						
11	Safety					
12						

Figure 9. An Empty Risk Matrix

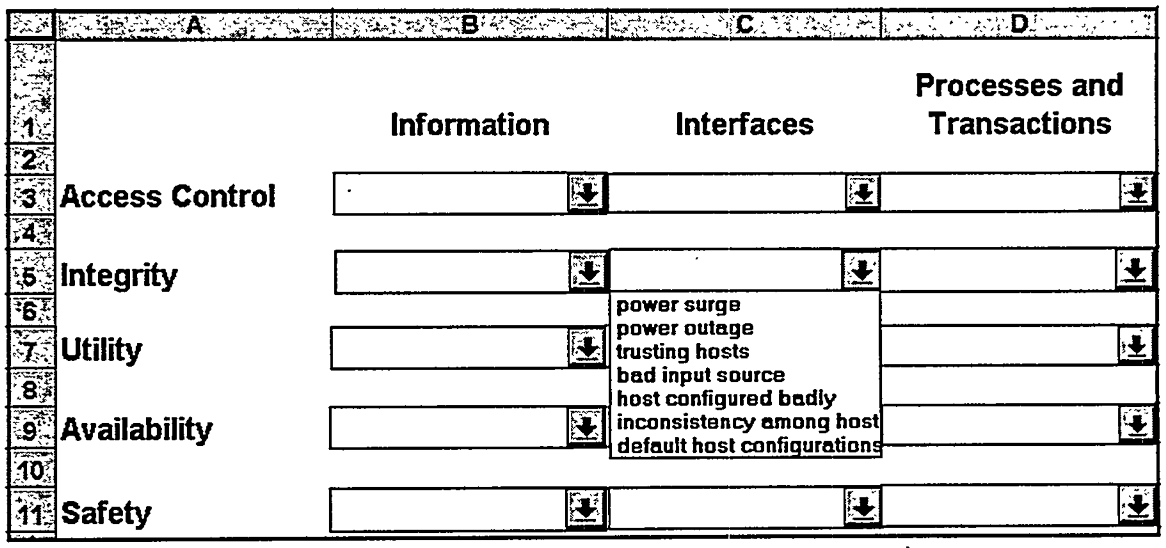
The risk specification matrix is used to select the sources of risks applicable to the information system at hand. This matrix has a drop down list box for each intersection of system aspect and surety objective (Figure 10).


A22

	Information	Interfaces	Processes and Transactions
Access Control			
Integrity			
Utility			
Availability			
Safety			

Figure 10. A Risk Specification Matrix

When the user clicks the mouse on a drop-down list box, the risks that are potentially applicable to that system aspect/surety objective intersection are displayed. Figure 11 shows an example of one such drop down.

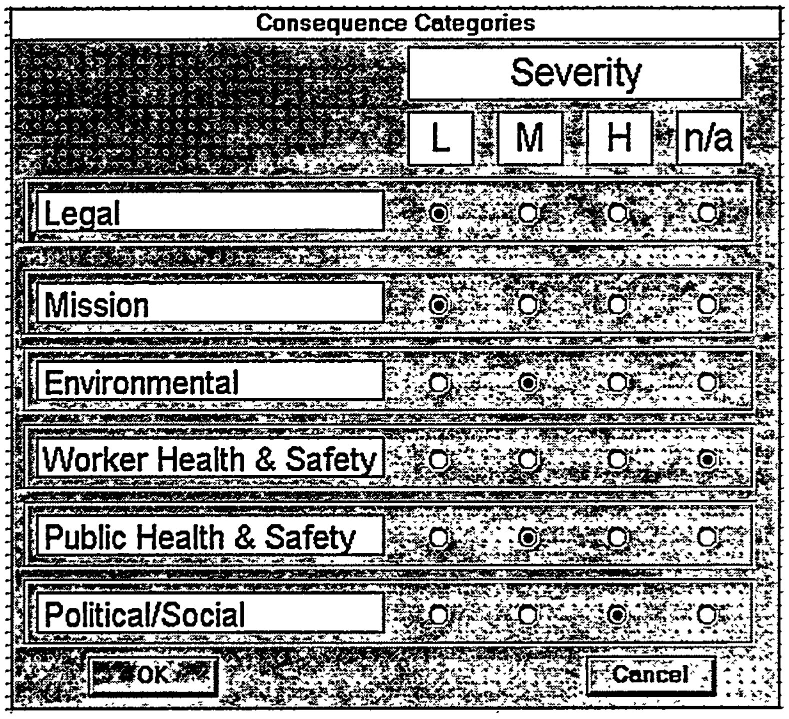

	Information	Interfaces	Processes and Transactions
Access Control			
Integrity			
Utility			
Availability			
Safety			

power surge
power outage
trusting hosts
bad input source
host configured badly
inconsistency among host
default host configurations

Figure 11. A Risk Specification Matrix with Sources of Risk Displayed

The list boxes in the risk specification matrix are populated with the risks from either the generic taxonomy or the system-specific risk taxonomy specified by the user, or both. The taxonomies from which the drop-down boxes are populated would be modified by the user via menu picks or button clicks on the toolbar; modifications would be saved to user-specified files.

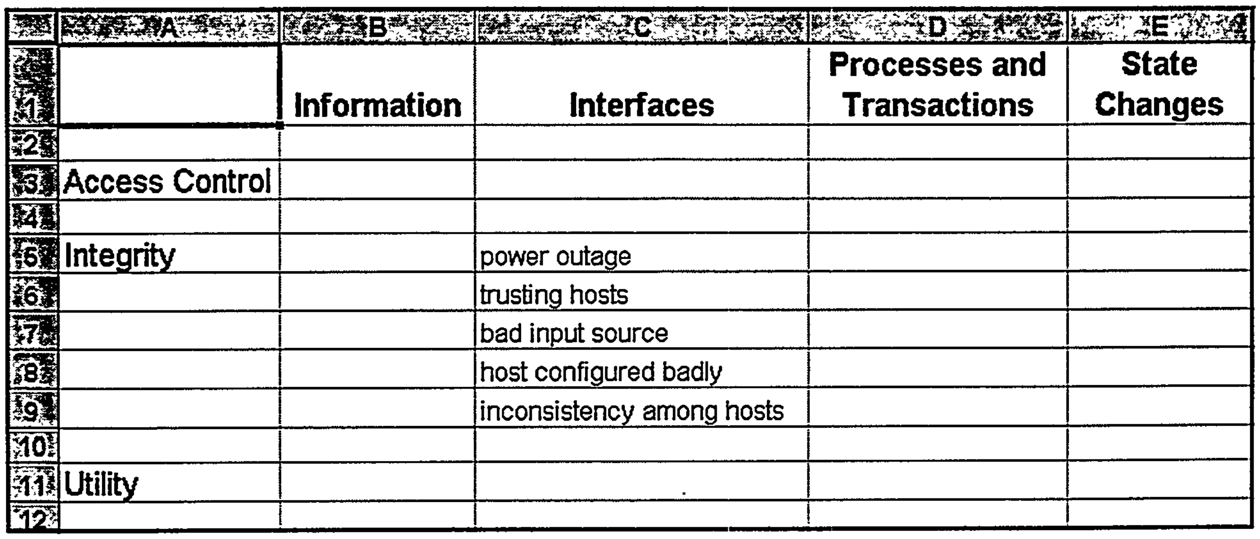
In addition to specifying what the risks sources are, there is a need to quantify the consequences if these risks are not mitigated, the area to which these consequences apply, the probability of occurrence, and the severity of the consequence. The dialog box shown in Figure 12 is used to elicit the severity information from the user. What to capture with respect to probability of occurrence and how to capture it has not yet been determined. The display of this dialog box requires the user to take some action.


Consequence Categories

	Severity			
	L	M	H	n/a
Legal	●	○	○	○
Mission	●	○	○	○
Environmental	○	●	○	○
Worker Health & Safety	○	○	○	●
Public Health & Safety	○	●	○	○
Political/Social	○	○	●	○

Figure 12. Consequence Area and Severity Assignment Dialog Box

Having selected the risks from the Risk Identification Matrix and specified their consequences (Figure 12), the risk matrix is populated with the specified risks (Figure 13)



	A	B	C	D	E
1		Information	Interfaces	Processes and Transactions	State Changes
2					
3	Access Control				
4					
6	Integrity		power outage		
6			trusting hosts		
7			bad input source		
8			host configured badly		
9			inconsistency among hosts		
10					
11	Utility				
12					

Figure 13. A Populated Risk Matrix

After the risks have been specified, mitigators need to be identified and their effectiveness quantified. Not all risks will necessarily require mitigators, although unmitigated risks may be of particular interest in the analysis results. Through any of various actions mentioned already (mouse clicks, menu picks, button selections or hot-keys) quantification of mitigators will be initiated. At this point in the development of the methodology and software, its existence and an estimate of its effectiveness are the only parameters quantifying a mitigator. In the future, quantification may include the mitigator's strength, its enforceability, and its type (purely procedural, purely technical, or a combination of both). Ideally, the user should be able to specify and quantify mitigators at any stage in the analysis of the system. Mitigators are captured in a matrix very similar to the risk matrix; the user chooses between the risks and mitigators by selecting one of the tabs in the lower left corner of the screen--System RISKS for the risk matrix, Risk MITIGATORS for the mitigators matrix (Figure 14).

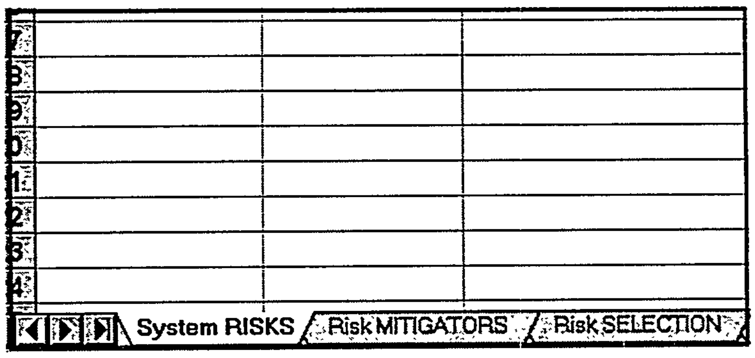


Figure 14. Risk Sources or Mitigators Selection Tabs

At this point, the software helps the user to think about and generate risk scenarios that could compromise system surety. The user is prompted for the sequence of events, states and actions that could lead from an initiating event to an undesired outcome. A series of questions asked of the user through dialog boxes is used to accomplish this task. These undesired outcomes are represented in the influence diagram with diamond-shaped symbols, and the text describing them is the logical negation of the system aspect to which the risk applies (e.g., lack of information integrity, lack of information utility, etc.) This set of events is one scenario or path through the influence diagram. There may be many paths from an initiating event to a value node. Generation of the scenarios or paths that lead to value nodes is not automatic; the user will be required to make a menu selection, push a button, etc. It would seem that depicting all of the individual paths produces an influence diagram that accurately and completely describes the information system. However, several iterations of the methodology will probably be necessary to discover hidden sources of risk and interactions among system components.

Having captured the risks, mitigators and sequence of events (scenario) leading to an undesired outcome, an influence diagram can now be created automatically. The reality, however, is that automatically collecting this information from a single user with no expectation of interaction with other human experts is a large, non-trivial task. For this reason, the created influence diagram should not be assumed to be complete and will require further iterations with subject matter experts for complete development. Figure 15 shows a section of one such influence diagram. The circles are risks, triangles are mitigators, diamonds are value nodes (undesired outcomes), and the numbers on the connecting lines are some measure of the residual risk in the system at that point.

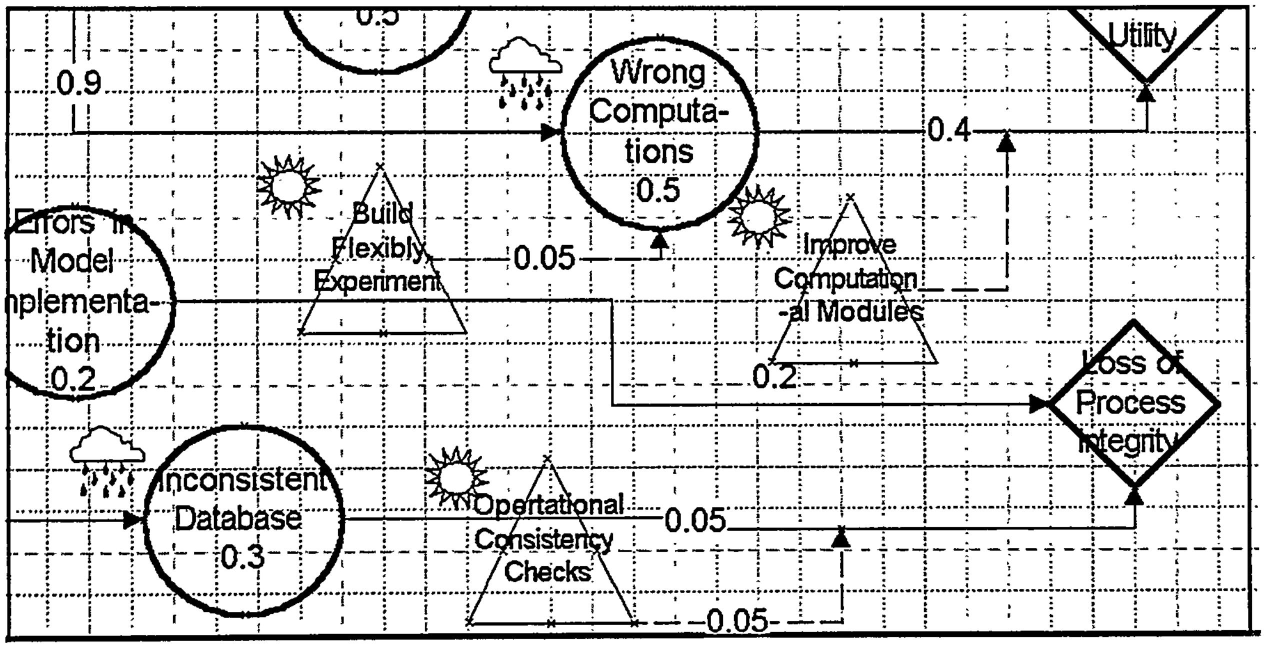

0.5
0.9
Utility
Wrong Computa-tions 0.5
0.4
Errors in Model Implementa-tion 0.2
Build Flexibly Experiment
0.05
Improve Computation-al Modules
0.2
Loss of Process Integrity
Inconsistent Database 0.3
Opertational Consistency Checks
0.05
0.05

Figure 15. A Partial Influence Diagram

The results of the analysis and how those results are presented could vary widely. However, the minimum set of information to be provided to the user would be the surety risk in the information system before any mitigation techniques have been applied, and the residual risk after the mitigators have been included. One way to state the risk remaining in the system is as a triplet consisting of the scenario or sequence of events leading to an end state (a path through the influence diagram), the probability of these events occurring, and the severity of the resulting consequences. The scenarios could be presented in various forms, although they should always be available in graphical form in the influence diagram, and highlighted or highlightable in some fashion (flashing path, bright colors, etc.). How best to present the risk triplet of scenario, probability, and severity is a subject still under investigation. Experience from the reactor safety arena indicates that presenting this information in a clear manner to those uninitiated in probabilistic risk assessment methods is not a trivial matter.

The possibilities for future development of the software are unbounded. However, we must determine who its users are and what would be most useful to that audience. For instance, other capabilities and information that some users might find desirable or useful include the following:

- the ability to query for consequences in a given category (legal, political, mission, etc.)
- the ability to query for consequences of a given probability of occurrence or a given severity level (currently specified as low, medium, high or not applicable for a given category)
- the ability to select a path (scenario) through the influence diagram
- mitigator parameterizations
- interactions of mitigators
- unmitigated risks
- outcomes of a specified state (scenario, consequences, probabilities and severity)

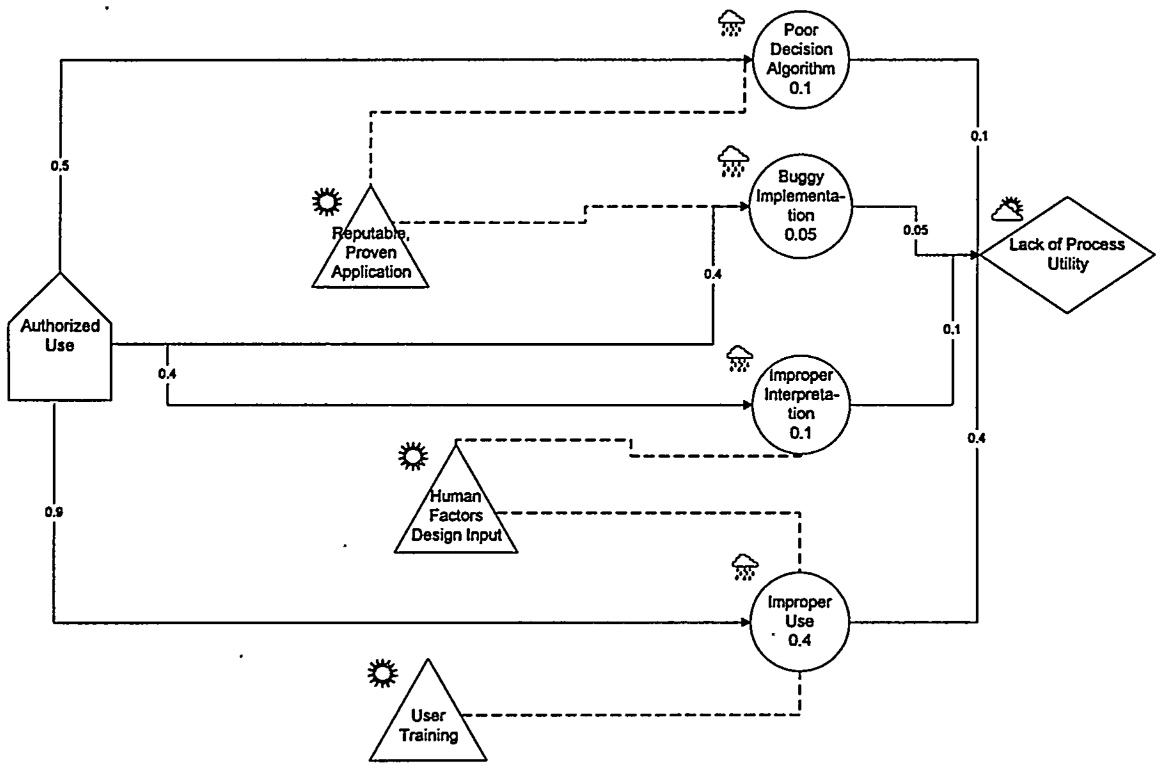
The software tool was created using all commercial off-the-shelf (COTS) software. The matrices were created in Microsoft Excel; the influence diagrams were created using ShapeWare's Visio drawing package; and the controlling macros, dialog boxes, drop down list boxes, etc. were created using one of the variations of Microsoft Visual Basic.

6 APPLICATIONS

We exercised various portions of the methodology using several applications, some of which have been used as examples in preceding sections of the paper. Project limitations prevented us from performing a complete risk assessment of a complex system, so we will use different examples to illustrate aspects of the methodology. To illustrate the influence diagram risk modeling technique, we will use as examples a medical decision support system and the risk assessment tool itself. To illustrate a method for eliciting information for constructing a risk matrix, we use an example of a firewall.

Medical Decision Support System Application. The Risk Identification Matrix in Figure 2 and the Risk Mitigators Matrix in Figure 3 represent some of the thought processes behind understanding what are the important sources of risk for a system using database information to facilitate medical treatment decisions. To begin construction of a graphical representation, we consider "utility risks relative to processes/transactions due to inappropriate processes." This source of risk is stated in the cell in Figure 2 where the surety objective "utility" intersects with the system aspect "processes/transactions." We have determined that the outcome of this risk has high consequences for our system because it may result in poor treatment decisions and health implications for the patient.

We will step through the risk assessment process using Figure 16 to illustrate.



Poor Decision Algorithm 0.1
Buggy Implementation 0.05
Improper Interpretation 0.1
Improper Use 0.4
Lack of Process Utility
Authorized Use
Reputable, Proven Application
Human Factors Design Input
User Training
0.5
0.4
0.4
0.9
0.1
0.05
0.1
0.4

Figure 16. Influence Diagram: Medical Decision Support System

1. Build system risk model, using input from matrices. The value node for the influence diagram is derived from the cell identifiers -- "lack of process utility." An initiating event would be "authorized use" -- simply, someone using the system. Working backward from the value node, we ask, "What could lead to lack of process utility?" and derive several risks, which translate into chance nodes. These more detailed risk sources may come from the Risk Identification Matrix, or they may be added to that matrix if they are developed in the process of constructing the graphical model.

(Adding them to the matrix allows development of a repository of risk sources.) Next, we consider mitigators, or barriers, for each risk. Again, these may come from the Risk Mitigators Matrix (e.g., "use a tested and proven process"), or they may be added to that matrix. In Figure 16, we see that "using a reputable, proven application" mitigates the risk caused by both "poor decision algorithm" and "buggy implementation." "Human factors design input" mitigates "improper interpretation" and "improper use," and "improper use" is also mitigated by "user training."

2. Analyze risk mitigators. We analyze the mitigators by considering the barrier characteristics illustrated in Figure 4, the Barrier Analysis Spreadsheet. We may use expert opinion, past experience, uncertainty analysis, etc., to derive probabilities for each chance node. (In fact, for our example, we simply made them up.) For example, the probability of a "poor decision algorithm," before mitigation, was determined to be 0.5, as shown on the arc between the initiating node and that chance node. After adding the barrier of using a "reputable, proven application," that probability goes down to 0.1, as shown in the chance node as well as on the arc from that node to the value node.
3. Run analysis engine. We may analyze the model as described in section 4.5.6, although with this simplistic example we can derive meaningful information by looking at the diagram.
4. Evaluate remaining risk; refine the analysis. We may decide that the barriers shown in the graph reduce all risks to acceptable levels except the risk of "improper use," which is at 0.4. So we may look for further mitigating techniques to apply to that source of risk. We then add to the model and matrix (Step 1), determine a probability for that barrier (Step 2), re-analyze the model (Step 3), and once again evaluate the remaining risk (Step 4).

Risk Assessment Tool Application. We also applied the methodology to the risk assessment tool itself and developed an influence diagram model illustrating some of the most important risks to that system. We considered that we are aiming to provide the software community a tool for risk management. Surely there are risks in trusting such a tool -- risk to the user's mission, and perhaps regulatory or social (embarrassment) risks as well -- should the tool mislead the user about system risks. Because we want the tool to be a sound and useful product, we decided to subject it to a risk analysis. Figure 17 shows the resulting system risk graph. Note that the high levels of concern around modeling, computation, and input are reflected in our basic project approaches, i.e., semantic descriptions, flexible tools, matrices serving as libraries, etc.

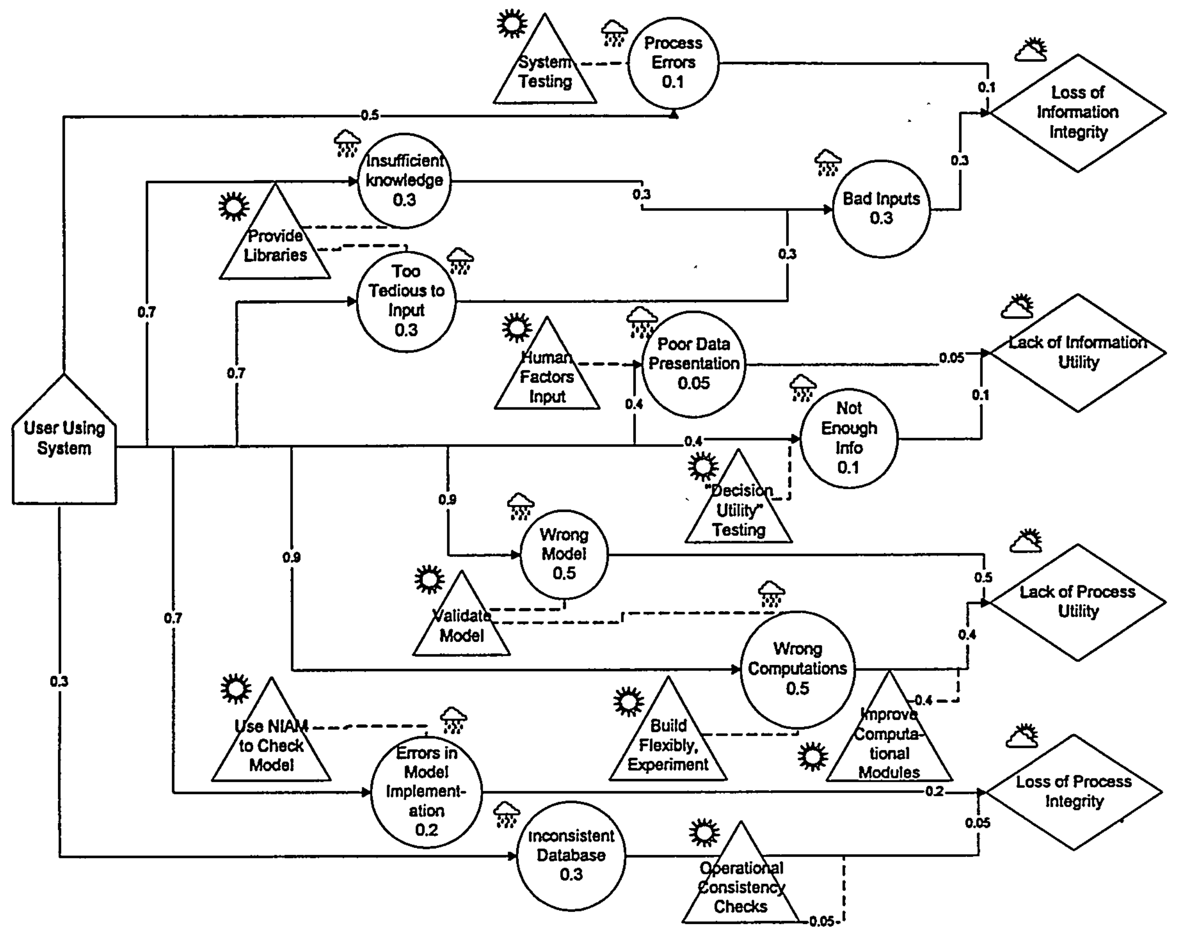

System Testing
Process Errors 0.1
Loss of Information Integrity
Insufficient knowledge 0.3
Provide Libraries
Too Tedious to Input 0.3
Bad Inputs 0.3
Human Factors Input
Poor Data Presentation 0.05
Lack of Information Utility
Not Enough Info 0.1
"Decision Utility" Testing
User Using System
Wrong Model 0.5
Validate Model
Lack of Process Utility
Wrong Computations 0.5
Improve Computational Modules
Build Flexibly, Experiment
Use NIAM to Check Model
Errors in Model Implementation 0.2
Loss of Process Integrity
Inconsistent Database 0.3
Operational Consistency Checks

Figure 17. Influence Diagram: Risk Assessment Tool

Firewall Application. We examined the information surety issues of a firewall application to exercise the Risk Identification Matrix as a guide for eliciting the broad range of risks to this type of information system. For each cell of the matrix, representing a surety objective in the context of an aspect of the system, we identified surety-related issues or questions. We found that the taxonomy lead us to consider areas that may not generally be considered as part of a firewall risk assessment, such as interface issues (administration and configuration of other machines on the network, availability of source code) and process issues (poor system design, poor software development, complexity). The amount of information quickly exploded, even though we had very limited access to a firewall expert to assist with the assessment.

The risk sources that we derived within the limitations of this project are listed in the Appendix. The list is organized using the taxonomy of the Risk Identification Matrix -- for each system aspect (column), potential risk sources are listed for each surety objective (row). We have no reason to believe that this list is complete; even so, it is obviously large. It is apparent that a completed software tool, such as the one we designed, is essential for organizing and tracking the information, evaluating severity of consequences for each risk source, identifying mitigators, evaluating mitigators, and performing tradeoffs, as well as actually constructing a diagram of varying levels of the system and its risks. A repository for accumulating this type of information, paired with the software tool for using appropriate subsets to analyze specific systems, would prove invaluable to analysts for understanding sources of risk and determining the best ways to mitigate them.

7 CONCLUSION

This report has described a research and development effort consisting of:

- an evaluation of current approaches and techniques for assessing risk in software systems;
- development of a methodology, technique, and tool to specify surety requirements and analyze a system's risks, taking advantage of appropriate aspects of existing methods and enhancing them where they are lacking;
- application of the methodology, technique, and tool to several types of information systems.

We described several existing techniques that can be used to assess information system surety. Our work has shown that, while each has its strengths, none of these techniques allows a holistic, risk-based approach to the assessment of information system surety. There is a need for a new method and language for expressing the surety requirements of a system and analyzing the remaining risk when the system is used for its intended purpose, and even for unintended purposes.

Our work led to the development of a method, technique, and tool to guide the analyst in the specification of system risk and analysis of the residual risk after barriers are introduced. It allows top-down and bottom-up development, allows several different system views (lists/matrices, process flows, graphical paths), encourages a system-specific view while providing a generic set of risks, and uses strong mathematical tools to determine residual risk. It is a probabilistic risk assessment type of tool based on the influence diagram technique, with enhancements to capture those components that were deemed lacking but necessary for complete system specification and analysis.

We applied the tool to several types of systems, including a robotic system, a medical decision support system, a firewall application, and the risk assessment tool itself. The results from applying the methodology to the tool itself led to many improvements to the methodology as it developed, especially the focus on flexibility, libraries, appropriate modeling techniques, and data presentation.

We were successful in developing a methodology with all of the characteristics we delineated in the early stages of the work (section 4.1)

- It avoids a compliance mentality and guides specification of actual surety objectives.

- It provides quantitative information that can be used to evaluate tradeoffs.
- It supports iterative refinement.
- It allows the analyst to focus on the most important issues, rather than forcing the inclusion of all risks.
- It combines the currently separate information system surety domains.
- The software facilitates model development, though project limitations prevented implementation of an automated technique for model solution.

The methodology also provides the analyst with assistance in the areas we had specified as essential:
- identifying system risks and surety requirements
- selecting mitigation techniques
- understanding and quantifying effectiveness, dependencies, and interactions of mitigators
- tailoring the model to the specific system

This approach is an important step in a new direction in the way we analyze information systems for robust application in today's rapidly changing computing environments. The interest it has generated when presented to various audiences (e.g., NIST Workshop on Dependable Systems, 1994; New Security Paradigms Workshop, 1995) is one indication of the utility of the method.

REFERENCES

Barroca, L. M., and J. A. McDermid, "Formal Methods: Use and Relevance for the Development of Safety-Critical Systems," *The Computer Journal,* Vol. 35, No. 6, 1992.

Bodeaù, D. J., and F. N. Chase, "Modeling Constructs for Describing a Complex System-of-Systems," *Ninth Annual Computer Security Applications Conference*, Orlando, Florida, December 6-10, 1993.

Camp, A. L., and L. P. Abeyta, "SANET 1.0 User's Guide and Reference Manual," SAND91-2864, Sandia National Laboratories, Albuquerque, New Mexico, 1991.

Common Criteria for Information Technology Security Evaluation, Common Criteria Editorial Board, Draft version 0.6, April 22, 1994.

Dobson, John, Centre for Software Reliability, University of Newcastle, personal communication, August, 1995.

Federal Criteria for Information Technology Security, Draft Version 1.0, December 1992.

Fortney, D. S., and J. J. Lim, "A Technical Approach for Determining the Importance of Information in Computerized Alarm Systems," *Proceedings of the 17th National Computer Security Conference*), Baltimore, MD, October 11-14, 1994.

Griesmeyer, J. M., and L. N. Smith, "A Reference Manual for the Event Progression Analysis Code (EVNTRE)," NUREG/CR-5174, Prepared by Sandia National Laboratories for the U.S. Nuclear Regulatory Commission, Washington, DC, 1989.

Howard, R. A., and J. E. Matheson, "Influence Diagrams," *Readings in the Principles and Applications of Decision Analysis*, p. 721, edited by R. A. Howard and J. E. Matheson, Strategic Decision Group, Menlo Park, CA., 1984.

Iman, R. L., and M. J. Shortencarier, "A FORTRAN 77 Program and User's Guide for the Generation of Latin Hypercube and Random Samples for Use With Computer Models," NUREG/CR-3624, Prepared by Sandia National Laboratories for the US Nuclear Regulatory Commission, Washington, DC, 1984.

Iman, R. L., and M. J. Shortencarier, "A User's Guide for the Top Event Matrix Analysis Code (TEMAC)," NUREG/CR-4598, Prepared by Sandia National Laboratories for the US Nuclear Regulatory Commission, Washington, DC, 1986.

Jae, M., and G. E. Apostolakis, "The Use of Influence Diagrams for Evaluating Severe Accident Management Strategies," *Nuclear Technology*, Vol. 99, pp. 142-157, 1992.

Jae, M., A. D. Milici, W. E. Kastenberg, and G. E. Apostolakis,, “Sensitivity and Uncertainty Analysis of Accident Management Strategies Involving Multiple Decisions,” *Nuclear Technology,* Vol. 104, pp. 13-36, 1993.

Leveson, N. G., and P. R. Harvey, "Analyzing Software Safety," *IEEE Transactions on Software Safety,* SE-9(5):569-579, September 1983.

Lim, J. J., An Analysis of the Risks of Data Aggregation via SecureNet, SAND95-8652, Sandia National Laboratories, CA, June 1995.

McDermid, J. A., “Issues in Developing Software for Safety Critical Systems,” *Reliability Engineering and System Safety,* 32 , 1991.

Nijssen, G. M., and K. W. Yunker, *Universal Informatics Workshop Material,* Nijssen Adviesbureau voor Informatica, The Netherlands, July 1993.

NIST, *Proceedings of the 4th International Computer Security Risk Management Model Builders Workshop,* Sponsored by NIST and the University of Maryland, August 6-8, 1991.

Parker, D., “Restating the Foundation of Information Security,” *Proceedings of the 14th National Computer Security Conference,* Washington, DC, October, 1991.

Parnas, D. L., et. al., “Evaluation of Safety-Critical Software,” *Communications of the ACM,* Volume 33, Number 6, June 1990.

Payne, A. C., Jr., and G. D. Wyss, “Coherent Sampling of Multiple Branch Event Tree Questions,” paper presented at PSAM-II, San Diego, CA, 1994.

Renis, T. A., R. A. Saleh and A. Sicherman, “Validating Detection Probabilities for the ASSESS Insider Database,” *Proceedings of the Institute of Nuclear Materials Management 31st Annual Meeting,* Los Angeles, CA, July 15-19, 1990.

Roberts, N. H., W. E. Vesely, D. F. Haasl, and F. F. Goldberg, “Fault Tree Handbook,” NUREG-0492, US Nuclear Regulatory Commission, Washington, DC, 1981.

Shachter, R. D., “Evaluating Influence Diagrams,” *Operations Research,* Vol. 34, p. 871, 1986.

US Nuclear Regulatory Commission (USNRC), “Severe Accident Risks: An Assessment for Five US Nuclear Power Plants,” NUREG-1150, US Nuclear Regulatory Commission, Washington, DC, 1990.

Wyss, G. D., and S. L. Daniel, "Recent Enhancements to Probabilistic Risk Assessment Software at Sandia National Laboratories," paper presented at the DOE EFCOG Integrated Risk Management Workshop, Albuquerque, New Mexico, 1994.

Wyss, G. D., et. al., "Toward a Risk-Based Approach to the Assessment of the Surety of Information Systems,' *1995 ASME Pressure Vessels and Piping Conference,* Honolulu, Hawaii, July 24-29, 1995.

PUBLICATIONS, OTHER PAPERS, PRESENTATIONS

Fletcher, S. K., "The Risk-Based Information System Design Paradigm," *Proceedings of the IFIP SEC'94 Conference*, May 1994.

Watterberg, P. A., S. K. Fletcher, R. D. Halbgewachs, R. M. Jansma, J. J. Lim, P. D. Sands, and G. D. Wyss, "Building a Surety Evaluation Model," submitted (but not accepted) to 17th National Computer Security Conference, 1994.

Fletcher, S. K., "The Case for a New Discipline: Information System Risk Management," presented at NIST ATP Workshop on Dependable Systems, 1994.

Lim J. J., S. K. Fletcher, R. D. Halbgewachs, R. M. Jansma, P. D. Sands, P. A. Watterberg, and G. D. Wyss, "Can Information be Assessed with High Confidence?," Poster Paper, High Consequence Operations Safety Symposium, July 1994.

Fletcher, S. K., R. D. Halbgewachs, R. M. Jansma, J. J. Lim, M. Murphy, P. D. Sands, and G. D. Wyss, CHISSA White Paper, January 1995.

Jansma, R. M., S. K. Fletcher, R. D. Halbgewachs, J. J. Lim, M. Murphy, P. D. Sands, and G. D. Wyss, "Risk-Based Assessment of the Surety of Information Systems," Eleventh International Symposium on the Creation of Electronic Health Record Systems and Global Conference on Patient Cards, March 1995.

Wyss, G. D., S. K. Fletcher, R. Halbgewachs, R. Jansma, J. Lim, M. Murphy, and P. D. Sands, "Towards a Risk-Based Approach to the Assessment of the Surety of Information Systems," American Society of Mechanical Engineers Pressure Vessels and Piping Conference, Topical Meeting "Risk and Safety Assessments: Where Is the Balance?," July 1995.

Fletcher, S. K., "A Methodology for Reasoning About Information System Surety," A Proposal Abstract, BAA95-15, March 1995.

Wyss, G. D., "Incorporating a Risk Perspective into Information System Surety," presented by S. K. Fletcher at the Energy and Environment Sector's Information Technologies Day at Sandia National Labs, March 6, 1995.

Fletcher, S. K., R. D. Halbgewachs, R. M. Jansma, J. J. Lim, M. Murphy, and G. D. Wyss, "Software Systems Risk Management and Assurance," New Security Paradigms Workshop, San Diego, CA, August 1995.

Fletcher, S. K., R. M. Jansma, J. J. Lim, M. Murphy, and G. D. Wyss, "Understanding and Managing Risk in Software Systems," Computer Security Applications Conference, New Orleans LA, December 1995.

Fletcher, S. K., R. M. Jansma, J. J. Lim, M. Murphy, and G. D. Wyss, "Managing Risk in Software Systems," paper submitted to IFIP '95.

APPENDIX

FIREWALL RISK TAXONOMY

INFORMATION

ACCESS CONTROL – INFORMATION

There is an Access Control risk relative to Information due to:

Access to the firewall hardware/software/data:

Are all invalid codes that can reach the firewall checked for? Or, could information come to the firewall which should cause an error condition have an unintended consequence to the system? Could this lead to an access control risk?

Can an unauthorized user/node impersonate a real or even privileged user/node? Can this be detected or responded to?

When critical data (e.g., passwords to log into firewall-related servers) is transferred over the network, can it be listened to? If it can, is that data useful to someone wanting to gain access to these machines?

Was a "back door" account left open by default, by an insider, or by an unauthorized change to the information in the account database?

Can information from a packet being transferred (or even evaluated for transfer) by the firewall affect the firewall itself in such a way as to allow access to the firewall hardware, software, or data (e.g., routing tables)?

Access to the internal network by unauthorized packets from the outside:

Can the information in a packet spoof the network into letting unauthorized outside traffic in or letting inside traffic out to unauthorized destinations?

Can information from the outside get through the firewall and cause a machine on the inside to grant the outsider "legitimate" access to the internal network (e.g., commands in a mail message)?

INTEGRITY – INFORMATION

There is an Integrity risk relative to Information due to:

System configuration integrity

Can information in the packets being passed or evaluated for passage cause a state change in the firewall?

Can the volume of information being processed by the firewall induce a state change?

Can this state change affect system configuration integrity?

Integrity of volatile data

Is there any way that the content of the information being processed can threaten the integrity of other stored data? (e.g., overwrite the wrong RAM area)

Can a high volume of information to be processed threaten the integrity of volatile data? (Buffer overruns, etc.)

Program (instruction) integrity

Can a high volume of information to be processed threaten the integrity of program instructions? (say, a buffer overruns and data overwrites program segments in volatile memory, or disk caching of packets overwrites program segments on a system disk)

Can the content of information packets being processed cause either of these?

Can legitimately issued (although errant or malicious) instructions from an authorized firewall administrator (user) cause either of these? Easily?

Permanent data integrity

See "Program (instruction) integrity" above as the same causes apply here.

Integrity of the data packets being transferred through the firewall

Does the firewall perform any intentional translation that might affect the integrity of the data being transferred through it? (translation doesn't work right, or is incompatible with the downstream use of the data)

Can a software or hardware fault cause the packet to be garbled as it passes through the firewall?

UTILITY – INFORMATION

There is a Utility risk relative to Information due to:

Utility: passing the packets that are supposed to get through

Can the volume of information to be processed by the firewall cause it to fail to pass some appropriate packets? Fail to pass them in a timely manner?

Can the information content in a legitimate packet cause the firewall to become confused and not pass the packet? (packet format, address, suspicious contents, etc.)

Also, see "Availability" below.

Utility: keeping out packets that are not supposed to get through

See "Access Control" above.

AVAILABILITY – INFORMATION

There is an Availability risk relative to Information due to:

State changes cause at least momentary unavailability. Can information in the packets being passed or evaluated for passage cause a state change in the firewall?

Can the type or content of the information in the packets being passed (or being evaluated for passage) cause the firewall to refuse to pass further packets (i.e., "shut down" or fail)?

Can the firewall be induced by information to go into a saturated or overload state? That is, can the volume of information flowing through the firewall impact its availability? Does the firewall handle the situation gracefully?

PROCESSES/TRANSACTIONS

ACCESS CONTROL – PROCESSES/TRANSACTIONS

There is an Access Control risk relative to Processes/Transactions due to:

authentication:
- based on believing the IP address
- reusable passwords
- plain text passwords on net (need "advanced authentication" techniques)

implementation:
- access controls implemented poorly

administration: (or is this State Changes?)
- bad system admin -- overworked, undertrained
- administering/maintaining complex firewall configuration
- patches
- configuration errors
- communication between sys. admins and firewall, security guys

spoofing:
- spoofing IP addresses -- IP source routing allowed (specify route to and from server; dynamic ARP, spoof client that is turned off)
- spoofing origin of email

testing:
- router filtering rules -- complexity and testing

other:
- unnecessary applications on firewall
- unnecessary accounts on firewall machines

INTEGRITY – PROCESSES/TRANSACTIONS

There is an Integrity risk relative to Processes/Transactions due to:

complexity:
- administering/maintaining complex firewall configuration
- number of systems -- greater #, greater risk
- router filtering rules -- complexity and testing

logs:
- level of protection of log files
- usability of reporting format
- inability to detect/stop attacks

other:
- malicious software
- human executes wrong process

UTILITY – PROCESSES/TRANSACTIONS

There is a Utility risk relative to Processes/Transactions due to:

design issues:
- poor system design
- poor software development

authentication:
- based on believing the IP address
- reusable passwords
- plain text passwords on net (need "advanced authentication" techniques)

implementation:
- access controls implemented poorly

complexity:
- administering/maintaining complex firewall configuration
- number of systems -- greater #, greater risk
- router filtering rules -- complexity and testing

administration: (or is this State Changes?)
- bad system admin -- overworked, undertrained
- administering/maintaining complex firewall configuration
- patches
- configuration errors
- communication between sys. admins and firewall, security guys

<u>spoofing</u>:

spoofing IP addresses -- IP source routing allowed (specify route to and from server; dynamic ARP, spoof client that is turned off)
spoofing origin of email

<u>testing</u>:

router filtering rules -- complexity and testing

<u>logs</u>:

level of protection of log files
usability of reporting format
inability to detect/stop attacks

<u>other</u>:

too many privileges for system operator
faulty applications (e.g. sendmail)
process error; unexpected action
unnecessary network services

AVAILABILITY – PROCESSES/TRANSACTIONS

There is an Availability risk relative to Processes/Transactions due to:

<u>complexity</u>:

administering/maintaining complex firewall configuration

<u>system overload:</u>

stupidity, malice, insufficient disk space, . . .

SYSTEM COMPOSITION

ACCESS CONTROL – SYSTEM COMPOSITION

There is an Access Control risk relative to System Composition due to:

<u>Access control design faults</u>

Mismatched implementation of policy among system elements
Open windows
Access control info subject to sniffing/eavesdropping/unauthorized access
Policy doesn't map to platform capabilities

<u>Operator/Administrator/User errors</u>

Unsynchronized configuration management among system elements
Unclear operational processes

Changes made without sufficient testing

<u>Networking makes system accessible to hackers</u>

INTEGRITY – SYSTEM COMPOSITION

There is an Integrity risk relative to System Composition due to:

<u>Configuration management unsynchronized among elements</u>
<u>Platform lacks protective mechanisms</u>
<u>Bad data driving system</u>

Database (router tables, etc) updates not synchronized

UTILITY – SYSTEM COMPOSITION

There is a Utility risk relative to System Composition due to:

<u>Poor task allocation between user & system</u>
<u>System vulnerable to jamming</u>
<u>Unsynchronized configuration management among elements</u>
<u>Architecture is limiting</u>

Some element limits total system utility
Elements cannot be synchronized

AVAILABILITY – SYSTEM COMPOSITION

There is an Availability risk relative to System Composition due to:

<u>Single point of failure</u>
<u>Outage</u>
<u>Design lacks fault tolerance</u>
<u>Transmission errors</u>

OTHER RISKS – SYSTEM COMPOSITION

<u>operating systems</u>:

security of firewall O/S
insecure default host configurations
patches and bug fixes

<u>user-controlled security:</u>

.rhosts
user-selected passwords
users adding modems
remote users -- through authorized modems only, using "advanced authentication"

insider downloading viruses

<u>inherent vulnerabilities in servers (?)</u>
<u>connections closing when they are supposed to; time out</u>

STATE CHANGES

ACCESS CONTROL – STATE CHANGES

There is an Access Control risk relative to State Changes due to:

<u>Access to the firewall hardware/software/data:</u>
Is physical protection of the system maintained during power outages?
Can a state change allow someone to gain privileged access (e.g., root)?
During a state change, is critical or privileged data passed across the network between the parts of the firewall system? Could it be viewed by inappropriate persons?
unauthorized access during:
disaster recovery
maintenance
testing
configuration changes
initial setup

<u>Access to the internal network by unauthorized packets from the outside:</u>
See "Utility" below.

INTEGRITY – STATE CHANGES

There is an Integrity risk relative to State Changes due to:

<u>System configuration integrity</u>
Power glitch damages one or more components
Parts of the system lose connectivity with one another
"Real time" problems (loss of synch, time-outs, etc.)
"Domino effect" - one failure causes the failure of other components -- software or hardware failure

<u>What volatile data is lost during a state change (say, reboot or power-down)?</u>
Possible gaps in audit trail (anything not committed to permanent storage)
Revert to out of date program or data on re-boot if program not properly saved

<u>Permanent data integrity</u>
Failure to properly close files - ungraceful state change corrupts files

Power glitch scrambles CMOS (system configuration data)

Program (instruction) integrity
Program storage damaged or overwritten by non-graceful state change

Integrity of the data packets being transferred through the firewall
Packets may be dropped or garbled by non-graceful state change

incorrect/inadequate implementation during:
disaster recovery
maintenance
testing
configuration changes
initial setup

UTILITY – STATE CHANGES

There is a Utility risk relative to State Changes due to:

Utility: passing the packets that are supposed to get through
Reboot: takes time (system can not pass any packets during reboot) - causes delay
Any state change may require the loading of new software or data - causes delay
Packets may be dropped or garbled by non-graceful state change
If continuity is not protected through a state change, may forget about existing sessions - require that they be re-established (annoying)

Utility: keeping out packets that are not supposed to get through
If continuity is not protected through a state change, may be spoofed into thinking an unauthorized packet belongs to a non-existent pre-state-change session
Is the list of restricted sites (inside to outside or outside to inside) maintained undamaged through the state change? Can it be subjected to unauthorized modification or spoofing as a result of a non-graceful state change?

incorrect/inadequate implementation during:
disaster recovery
maintenance
testing
configuration changes
initial setup

AVAILABILITY – STATE CHANGES

There is an Availability risk relative to State Changes due to:

<u>Obvious state change</u>:
- shutdown (planned or unplanned) incapacitates the system

<u>Reboot</u>:
- takes time (system can not pass any packets during reboot) - causes delay
- Any state change will likely require the loading of new software or data - causes delay

<u>firewall is down during</u>:
- disaster recovery
- maintenance
- testing
- configuration changes
- initial setup

OTHER RISKS – STATE CHANGES

<u>maintenance/upgrades/changes:</u>
- process for installing patches, fixing vulnerabilities that are discovered
- flexibility of the firewall wrt policy, functionality changes

---

INTERFACES

ACCESS CONTROL – INTERFACES

There is an Access Control risk relative to Interfaces due to:

<u>administration:</u>
- admin of other site machines (the firewall doesn't do it all; and what if it's penetrated)

<u>source code available for bad guys to study</u>
<u>different surety policies across interfaces</u>
<u>compromise of:</u>
- privileged account on firewall
- router configuration
- network database

INTEGRITY – INTERFACES

There is an Integrity risk relative to Interfaces due to:

hosts on network: (or is this System Composition?)
host configured badly; inconsistency among hosts
trusting hosts
default host configurations insecure

power surge
power outage
bad input source
compromise of:
proxy gateways to firewall (telnet, X gate, mail gate, name server)
authentication mechanism (e.g., Kerberos serer)
security policy (e.g. users tunneling through firewall)

UTILITY – INTERFACES

There is a Utility risk relative to Interfaces due to:

administration:
admin of other site machines (the firewall doesn't do it all; and what if it's penetrated)

back door:
network connections that don't go through firewall
users adding modems
dial-out modems
SLIP, PPP connections
is modem service effective so they won't be tempted to cheat?

hosts on network: (System Composition?)
host configured badly; inconsistency among hosts
trusting hosts
default host configurations insecure

policy:
firewall policy incorporated into overall site policy; and all the other policy stuff -- is there one, is it clear, was it developed before the system . . .; criteria for adding functionality to firewall

incidents:
incident response plans
detection/response/recovery

other:
source code available for bad guys to study
bad source information -- network database, name server

<u>compromise of:</u>

- proxy gateways to firewall (telnet, X gate, mail gate, name server)
- authentication mechanism (e.g., Kerberos serer)
- security policy (e.g. users circumventing firewall via other paths)

AVAILABILITY – INTERFACES

There is an Availability risk relative to Interfaces due to:

<u>power outage</u>
<u>power surge</u>
<u>A/C failure</u>
<u>network down</u>
<u>interfacing machines/services down</u>:

- Kerberos server
- network database
- proxy gateways

OTHER RISKS

eavesdropping:
unencrypted email
passwords on network

threat agent motivation:
level of attraction to an intruder

DISTRIBUTION:

5	MS	0449	Sharon Fletcher, 9411
5		0449	Martin Murphy, 9411
10		0451	Roxana Jansma, 9415
5		0747	Gregory Wyss, 6412
1		0877	Ronald Halbgewachs, 5931
1		0806	Paul Sands, 4621
1		1008	Peter Watterberg, 9621
1		0451	Judy Moore, 9415
1		0780	Tricia Sprauer, 5838
1		0319	Gary Randall, 2645
1		1436	LDRD Office, 4523
1		9018	Central Technical Files, 8523-2
5		0899	Technical Library, 4414
2		0619	Review and Approval Desk, 12630 For DOE/OSTI