SAND96-1140C

CONF-960869--5

Risk Management - What About Software?

Sharon K. Fletcher, Ph.D.; Sandia National Laboratories, Albuquerque NM

mailing address:
S. K. Fletcher
MS0449
Sandia National Laboratories
Albuquerque, NM 87185-0449

email:
skfletc@sandia.gov

phone: (505)844-2251
fax: (505)844-9641

RECEIVED
JUN 03 1996
OSTI

This work was performed at Sandia National Laboratories, which is operated by Lockheed-Martin Corporation for the U.S. Department of Energy under contract DE-AC04-94-AL85000.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

(draft paper for peer review)

Risk Management - What About Software?

(author)

Abstract

Risks in software systems arise from many directions. There are risks that the software is faulty, that the system may be attacked, that safety hazards exist, that the system may be inoperable or untimely, that an abnormal event may cause unexpected actions, etc. Risk analysis tools should support and document risk-mitigation decisions, and facilitate understanding of residual risks. These tools must be based on a sound theory of risk, which does not exist today. Probabilistic risk assessment techniques apply to physically-based systems where failure modes and event dependence are fairly well understood. But they cannot be blindly applied to software systems, which do not share these characteristics. Moreover, we need to meld many diverse aspects of risk for software systems. This presentation will explore some thought-provoking ideas about modeling, problem spaces, solution approaches, math, decision friendly output, and the role of risk analysis in the software lifecycle.

Introduction

Risk management is a well established concept in many fields, but software is not one of them. Software risk management is very much in its infancy, and needs some direction in order to evolve into a usable science. Most will agree that software is a unique kind of animal. Its uniquenesses have important implications for developing risk analysis and decision support tools.

Adopting a Viewpoint and Defining Risk

In a software system, risk can have many disparate sources - faults, errors, hazards, abnormal events, unexpected environments, attacks, untimeliness, unavailability, the system development process, operational procedures, maintenance, etc. Within the software community, separate interest groups have formed to address some of these risks, although not all have thought of their jobs as risk management. These include security, safety, dependability, and development processes. Within each of these areas, there are even more specialized interests, such as multi-level-security, communications security, asset protection, hazops, first principles, fault tolerance, database integrity, process maturity, testing, and configuration management. The words "risk management" conjure up very different ideas within these different interest groups.

Software's first uniqueness is that there is no obvious correct viewpoint

Any single focus from the above list is clearly inadequate. Choosing a viewpoint on the problem is critically important, for the problem viewpoint restricts the solutions that one is able to see. Perhaps the most basic and encompassing viewpoint to take is that of **correct system operation**, achievable through an appropriate balance of all other concerns. This viewpoint can span the entire lifecycle, including the processes used for development, operation, and maintenance. It can also span all aspects of the system that might contribute to risk, such as its architecture, functions, information, interfaces, and environment. The "risks" to be managed can be described in terms of failure to achieve and maintain the appropriate balance of concerns, or "surety objectives", for correct system operation.

Risk = the possibility of not maintaining a surety objective

What has been said so far for software systems could be said for any system. But one of the unique things about software is the strong interdependence of surety objectives. At the high level, it is easy to see that safety requires dependability, security impacts availability, and these things would be true for any system. What seems to be especially troublesome in software is the degree to which these objectives become all mixed up and inseparable in a single implementation of a system. There is heavy interaction among risk mitigators; that is, measures applied to one objective will frequently impact others as well. Communications security is a boon for data integrity, but a problem for timeliness and availability. Theoretically, a fail-over design provides fault tolerance which improves correctness, but by its very complexity may actually have the opposite effect. Access controls protect system integrity, but may impede access to critical data in a crisis. A server architecture enhances integrity by avoiding duplicated data, but may introduce a single point of failure. The tradeoffs seem endless. And traceability is tough. Software, perhaps more than any other domain, suffers from inseparability of surety objectives.

Software's second uniqueness is inseparability of surety objectives

Understanding the Needs of the Software Community

Risk management and decision support tools for the software community must help the system developer to identify, document, and balance risk mitigation over all surety objectives. Each of these is explored in more detail below.

The first need is to identify risks. Any methodology for identifying risks in software systems should encompass all surety objectives, cover the entire lifecycle, and include operational dynamics. There are partial methodologies in use today which emphasize one or another surety objective, and usually only one portion of the system lifecycle. For example, a security risk analysis for operating system selection based on confidentiality levels of data and users, a software development risk analysis yielding process improvements for quality-schedule-budget, a safety risk analysis yielding system design decisions (typically starting with a clean sheet of paper for each safety risk to be considered, and not dealing with the interactions), a disaster recovery risk analysis yielding operational contingency plans, a transaction-oriented risk analysis emphasizing checkpoint-and-restart design to maintain data integrity, and a physical security risk analysis emphasizing protection of assets.

The thought of covering all these bases in a single risk analysis is daunting, but it can be made manageable with good tools and iterative refinement. The practical cost of not doing so is heavy and often the downfall of entire projects. By not considering and prioritizing the full breadth of risks, developers waste resources addressing the wrong problems while the unrecognized real problems doom the system or project to failure. It is human nature to solve what we know how to solve, measure what we know how to measure, and build what we've built before, whether or not it is what we need now.

The second need is to document risks and their mitigation. Documentation forms the basis for certification and accreditation decisions. Even so, the rationale for surety approaches is typically not well recorded. Risk mitigation often involves a combination of technology and process, and a very strong technological solution can be undermined by lax procedures. Additionally, it is important to understand original intents so they are not violated by future changes. This brings out another important and unique characteristic of software: maintenance means change. The heavy interaction of risks and mitigators in software systems, discussed above, can give rise to vulnerabilities if some mitigator is replaced without recognizing that it also had a secondary role in mitigating other risks.

Software's third uniqueness is that maintenance means change

The third need is to balance risks. Balancing happens in several contexts. When surety objectives compete, mitigating risk in one raises risk in the other, so a balance must be found in which both are mitigated to an acceptable level. When mitigating a single risk involves many mechanisms in many parts of the system, balance means avoiding the strong door-weak window syndrome. Sometimes balance means selecting the best mix of technology and procedure, or physical and software controls. Balance can also mean cost/benefit considerations.

Theories of Risks

Probabilistic risk assessment (PRA) is a well established field, whose approaches are routinely applied to critical systems applications such as nuclear reactors. PRA consists of a suite of methodologies, using trees, graphs, tables, or block diagrams to explore causes and effects. Once a system under study is modeled with a tree or other construct, the model can be "solved" with the mathematics of probabilities.

A drawback to block diagrams is that they tend to adopt one narrow view of the system, such as physical layout or process flow. Software's first uniqueness tells us this will not do. A drawback to table-based approaches is that they tend not to deal with interactions across components or events, which is equivalent to assuming total independence. Software's second uniqueness tells us this will not do. In the tree and graph techniques, risk quantification is based on conditional probabilities of combinations and series of events leading up to undesirable events. The mathematics used to combine probabilities assumes simple (ands and ors) interactions of events, and probabilities that do not vary over time. PRA is typically applied to assessing component failures in systems where these assumptions (this "theory of risk") hold. Software is not one of them.

However, the thought processes that go into probabilistic risk assessments are generally applicable to software. We can use them if we're careful not to blindly apply inappropriate mathematics and not to pull probabilities from the air with no basis. In fact, what may be needed is different "solution" methods for existing PRA approaches, i.e., a different "theory of risk" for software. But before we decide, we should also consider that the software field gives us several more ways of looking at things: data flow, control flow, state machines, Petri nets, Markov models, etc. And we should consider Nancy Leveson's admonition that software exhibits neither unorganized complexity nor organized simplicity, but rather organized complexity. This is a very important observation, for it tells us why we can never hope to analyze software systems with probabilities, nor with analytic reduction.

Software's fourth uniqueness is organized complexity

A Theory of Risk for Software Systems

Graph techniques are a good match for representing many things about software, as evidenced by the list in the previous paragraph. And a fairly recent innovation in the PRA arena, a graph technique called the Influence Diagram, is gaining popularity for its flexibility. Graphs also have visual appeal over trees because they eliminate redundancy and can give one the system-at-a-glance, especially when they are developed hierarchically via iterative refinement.

So, we select the graph. Now, what should the nodes and edges represent? The focus of correct system operation by **maintaining surety objectives**, suggests that paths through the graph could be system risk states that culminate in the negation of some surety objective. So, nodes will be risk states, and edges will represent transitions between states. Depending on the system at hand, the surety objectives of interest, and the analyst's inclination, the graph might resemble process or information flow, or a physical layout, but generally we wouldn't expect it to. Typically, it will be more abstract, depicting events or states of the system in its broadest context. Recalling that the user wants to identify, document, mitigate, and balance risks, we should add a capability to insert mitigators, which would block or lessen the possibility of state transitions. The analyst will want to experiment with alternative mitigators at various points along a path, and with layering them (stacking them up serially), and with tradeoffs to find acceptable balance. Figure 1 depicts a general graph with mitigators. In the figure, tables accompany each mitigator and accompany a threat which is being introduced into the graph. These tables represent characteristics of threats and mitigators that the analyst can assess to evaluate their relative strengths. For simplicity, this figure depicts mitigators on paths between states, but more generally mitigators should be thought of as standing off the paths and influencing multiple paths. This concept, which we borrowed from Influence Diagrams, is what allows capture of the interactions discussed earlier.

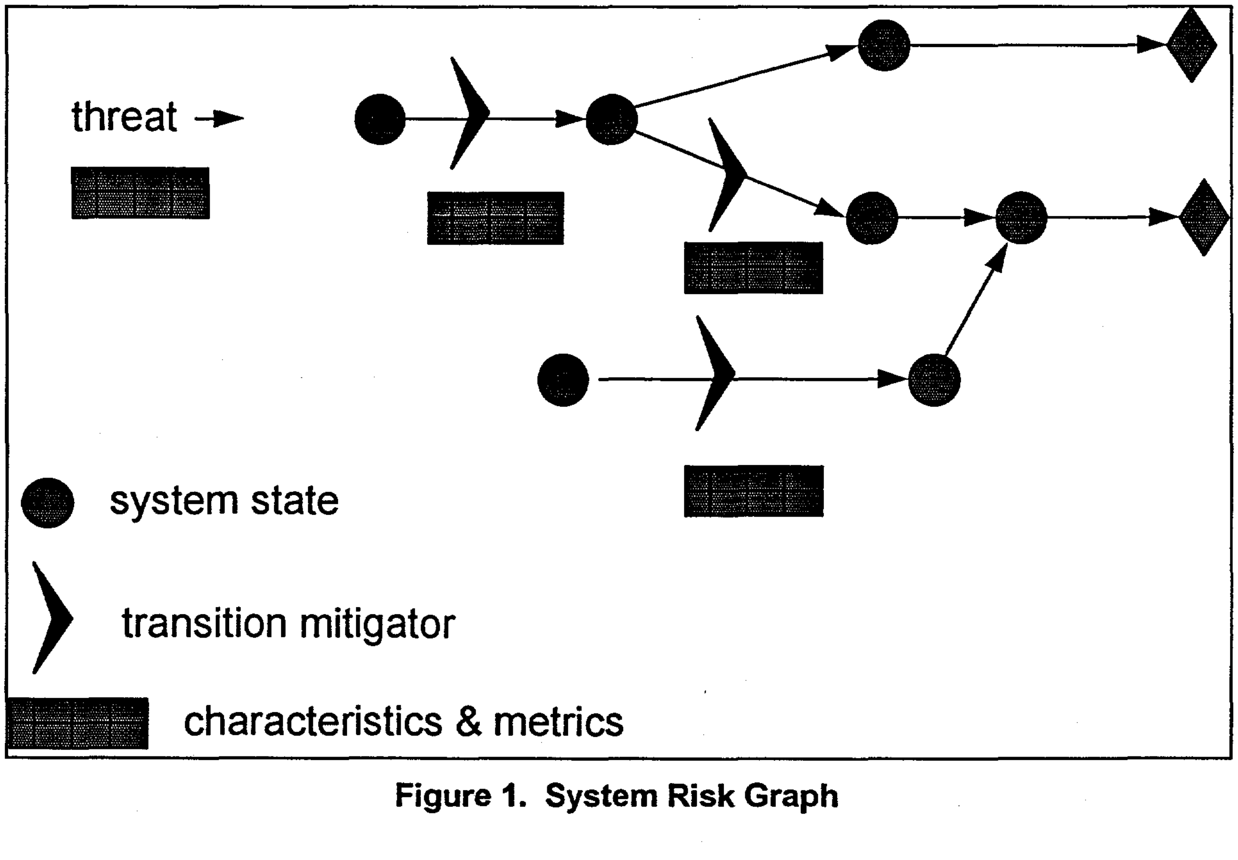

Figure 1. System Risk Graph

The picture is nice, and can stand on its own as a comprehensive and comprehensible qualitative risk assessment. But we are not yet to a theory of risk. The theory of risk is the function, the mathematics or logic, the calculations to be made over the graph to measure the risk reduction that can be achieved and the remaining residual risk. It is the model of how risk states, threats, and mitigators interact to push us towards or keep us from hitting the undesirable states. It includes the scales on which we measure these things, too. Here is where we must throw out the traditional PRA "solution methods." Instead, we seek a mathematical model that works for organized complexity, for things measured on different scales, and for data with wildly varying uncertainties.

We would be foolish to put precise mathematics to this today. As the field develops, data will be more certain and more precision will emerge; future developers will need to pay close attention to how uncertainties figure into calculations and to error accumulation in computations. Today, we should seek to only slightly quantify the qualitative assessment, perhaps with scales like L-M-H or 1-to-5. The interactions of threats, mitigators, and states might be "computed" via logic or decision tables, rather than formulas. The next step might be to investigate some of the newer branches of mathematics that take into account various sources of uncertainty - randomness, conflicting evidence, confusion, lack of information, etc. These mathematics include possibilistic, fuzzy, evidential, and Dempster-Scheaffer.

System Aspects / **Surety Obj**	Information	Transaction	Composition Architecture	State Chng	Interfaces
Access Control	• eavesdropping • A. C. failure	• spoofing • authent. failure	• passwords exposed on net • lacks features	• abnormal event • access by repair personnel	• software & system policies mismatched
Integrity	• unauthorized modification	• repudiation • subversion	• DBMS lacks integrity checks • untrained users	• incomplete updates due to maintenance	• incomplete input • bad source
Availability	• inappropriate access control • too slow	• overload • timing design fault	• single p.o.f. • communications sabotage	• natural disaster • sabotage • in maintenance	• power interruption
Utility	• unauth. modific. • accidental modification	• unplanned environment	• multiple copies on servers out of synch	• shutdown-startup not synchronized	• output misinterpreted
Safety	• incorrect data • insufficient information	• out of tolerance	• not a fail-safe design	• inappropriate response to abnormal event	• unchecked input

Figure 2. Software System Risk Matrix

Conclusions

The "theory of risk" development above is not applicable solely to software. It applies to any system that is characterized by organized complexity. In fact, software is always part of a larger system, and the boundaries of analysis can be set inside or outside the software portion. We have customized this approach to software by defining software-relevant surety objectives, and by identifying views of software systems that are conducive to identifying risk we are concerned with. This results in two axes of a matrix, shown in Figure 2. We carry out risk identification in the matrix, then map the results to the graphical form and insert the mitigators. Our analysis from that point is manual and very roughly quantitative, of the L-M-H sort. The primary benefits we have derived are: we think much more broadly about risks than we otherwise would, the graph documents our decisions about mitigators, interactions of mitigators are depicted, we can what-if to our hearts' content, and in the end we must consciously accept the residual risk laid out before us.

Biography

References

1. Leveson, Nancy, Safeware: System Safety and Computers, Addison Wesley, 1995.

2. Wyss, Gregory D., et. al., "Toward a Risk Based Approach to the Assessment of the Surety of Information Systems"

3. Lim, J. J., et. al., "Can Information Surety be Assessed with High Confidence?"

4. (math book)