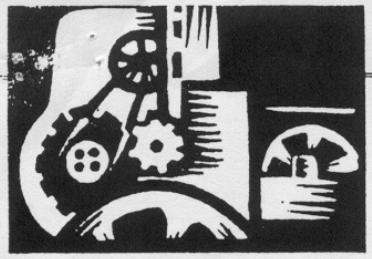


3

Software Risk Management: Principles and Practices

BARRY W. BOEHM,
Defense Advanced Research Projects Agency

◆ *Identifying and dealing with risks early in development lessens long-term costs and helps prevent software disasters.*
It is easy to begin managing risks in your environment.

Like many fields in their early stages, the software field has had its share of project disasters: the software equivalents of the Beauvais Cathedral, the *HMS Titanic*, and the "Galloping Gertie" Tacoma Narrows Bridge. The frequency of these software-project disasters is a serious concern: A recent survey of 600 firms indicated that 35 percent of them had at least one runaway software project.[1]

Most postmortems of these software-project disasters have indicated that their problems would have been avoided or strongly reduced if there had been an explicit early concern with identifying and resolving their high-risk elements. Frequently, these projects were swept along by a tide of optimistic enthusiasm during their early phases that caused them to miss some clear signals of high-risk issues that proved to be their downfall later.

Enthusiasm for new software capabilities is a good thing. But it must be tempered with a concern for early identification and resolution of a project's high-risk elements so people can get these resolved early and then focus their enthusiasm and energy on the positive aspects of their product.

Current approaches to the software process make it too easy for projects to make high-risk commitments that they will later regret:

♦ The sequential, document-driven waterfall process model tempts people to overpromise software capabilities in contractually binding requirements specifications before they understand their risk implications.

♦ The code-driven, evolutionary development process model tempts people to say, "Here are some neat ideas I'd like to put into this system. I'll code them up, and

0740-7459/91/0100/0032/$01.00 © IEEE

if they don't fit other people's ideas, we'll just evolve things until they work." This sort of approach usually works fine in some well-supported minidomains like spreadsheet applications but, in more complex application domains, it most often creates or neglects unsalvageable high-risk elements and leads the project down the path to disaster.

At TRW and elsewhere, I have had the good fortune to observe many project managers at work firsthand and to try to understand and apply the factors that distinguished the more successful project managers from the less successful ones. Some successfully used a waterfall approach, others successfully used an evolutionary development approach, and still others successfully orchestrated complex mixtures of these and other approaches involving prototyping, simulation, commercial software, executable specifications, tiger teams, design competitions, subcontracting, and various kinds of cost-benefit analyses.

One pattern that emerged very strongly was that the successful project managers were good *risk managers*. Although they generally didn't use such terms as "risk identification," "risk assessment," "risk-management planning," or "risk monitoring," they were using a general concept of risk exposure (potential loss times the probability of loss) to guide their priorities and actions. And their projects tended to avoid pitfalls and produce good products.

The emerging discipline of software risk management is an attempt to formalize these risk-oriented correlates of success into a readily applicable set of principles and practices. Its objectives are to identify, address, and eliminate risk items before they become either threats to successful software operation or major sources of software rework.

BASIC CONCEPTS

Webster's dictionary defines "risk" as "the possibility of loss or injury." This definition can be translated into the fundamental concept of risk management: risk exposure, sometimes also called "risk impact" or "risk factor." Risk exposure is defined by the relationship

RE = P(UO) * L(UO)

where RE is the risk exposure, P(UO) is the probability of an unsatisfactory outcome and L(UO) is the loss to the parties affected if the outcome is unsatisfactory. To relate this definition to software projects, we need a definition of "unsatisfactory outcome."

Given that projects involve several classes of participants (customer, developer, user, and maintainer), each with different but highly important satisfaction criteria, it is clear that "unsatisfactory outcome" is multidimensional:

- For customers and developers, budget overruns and schedule slips are unsatisfactory.
- For users, products with the wrong functionality, user-interface shortfalls, performance shortfalls, or reliability shortfalls are unsatisfactory.
- For maintainers, poor-quality software is unsatisfactory.

These components of an unsatisfactory outcome provide a top-level checklist for identifying and assessing risk items.

A fundamental risk-analysis paradigm is the decision tree. Figure 1 illustrates a potentially risky situation involving the software controlling a satellite experiment. The software has been under development by the experiment team, which understands the experiment well but is inexperienced in and somewhat casual about software development. As a result, the satellite-platform manager has obtained an estimate that there is a probability P(UO) of 0.4 that the experimenters' software will have a critical error: one that will wipe out the entire experiment and cause an associated loss L(UO) of the total $20 million investment in the experiment.

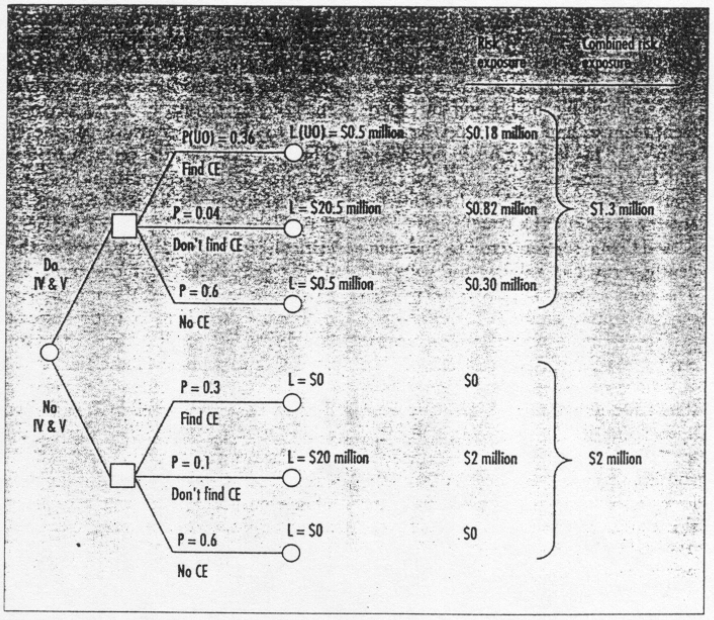


FIGURE 1. DECISION TREE FOR WHETHER TO PERFORM INDEPENDENT VALIDATION AND VERIFICATION TO ELIMINATE CRITICAL ERRORS IN A SATELLITE-EXPERIMENT PROGRAM. L(UO) IS THE LOSS ASSOCIATED WITH AN UNSATISFACTORY OUTCOME, P(UO) IS THE PROBABILITY OF THE UNSATISFACTORY OUTCOME, AND CE IS A CRITICAL ERROR.

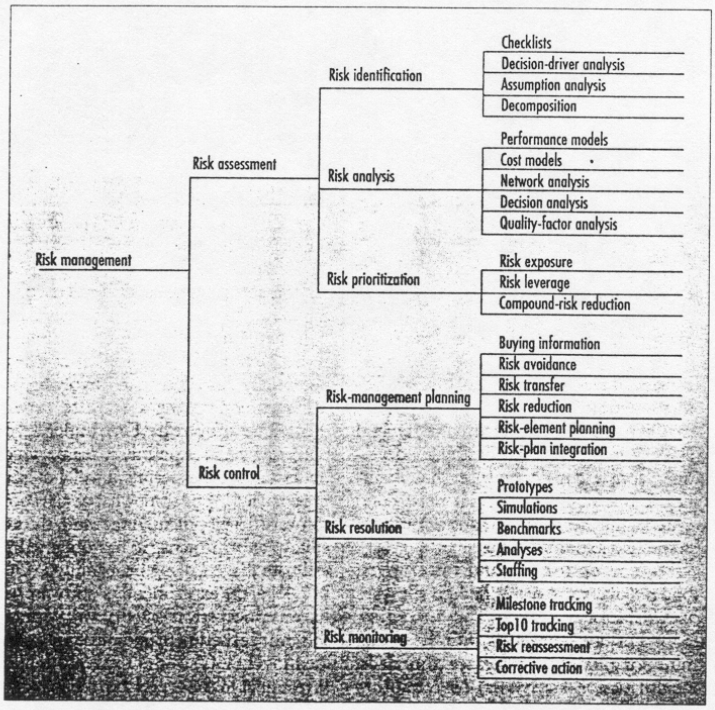


FIGURE 2. SOFTWARE RISK MANAGEMENT STEPS.

The satellite-platform manager identifies two major options for reducing the risk of losing the experiment:

- Convincing and helping the experiment team to apply better development methods. This incurs no additional cost and, from previous experience, the manager estimates that this will reduce the error probability P(UO) to 0.1.
- Hiring a contractor to independently verify and validate the software. This costs an additional $500,000; based on the results of similar IV&V efforts, the manager estimates that this will reduce the error probability P(UO) to 0.04.

The decision tree in Figure 1 then shows, for each of the two major decision options, the possible outcomes in terms of the critical error existing or being found and eliminated, their probabilities, the losses associated with each outcome, the risk exposure associated with each outcome, and the total risk exposure (or expected loss) associated with each decision option. In this case, the total risk exposure associated with the experiment-team option is only $2 million. For the IV&V option, the total risk exposure is only $1.3 million, so it represents the more attractive option.

Besides providing individual solutions for risk-management situations, the decision tree also provides a framework for analyzing the sensitivity of preferred solutions to the risk-exposure parameters. Thus, for example, the experiment-team option would be preferred if the loss due to a critical error were less than $13 million, if the experiment team could reduce its critical-error probability to less than 0.065, if the IV&V team cost more than $1.2 million, if the IV&V team could not reduce the probability of critical error to less than 0.075, or if there were various partial combinations of these possibilities.

This sort of sensitivity analysis helps deal with many situations in which probabilities and losses cannot be estimated well enough to perform a precise analysis. The risk-exposure framework also supports some even more approximate but still very useful approaches, like range estimation and scale-of-10 estimation.

RISK MANAGMENT

As Figure 2 shows, the practice of risk management involves two primary steps each with three subsidiary steps.

The first primary step, risk assessment, involves risk identification, risk analysis, and risk prioritization:

- Risk identification produces lists of the project-specific risk items likely to compromise a project's success. Typical risk-identification techniques include checklists, examination of decision drivers, comparison with experience (assumption analysis), and decomposition.
- Risk analysis assesses the loss probability and loss magnitude for each identified risk item, and it assesses compound risks in risk-item interactions. Typical techniques include performance models, cost models, network analysis, statistical decision analysis, and quality-factor (like reliability, availability, and security) analysis.
- Risk prioritization produces a ranked ordering of the risk items identified and analyzed. Typical techniques include risk-exposure analysis, risk-reduction leverage analysis (particularly involving cost-benefit analysis), and Delphi or group-consensus techniques.

The second primary step, risk control, involves risk-management planning, risk resolution, and risk monitoring:

- Risk-management planning helps prepare you to address each risk item (for example, via information buying, risk avoidance, risk transfer, or risk reduction), including the coordination of the individual risk-item plans with each other and with the overall project plan. Typical techniques include checklists of risk-resolution techniques, cost-benefit analysis, and standard risk-management plan outlines, forms, and elements.
- Risk resolution produces a situation in which the risk items are eliminated or otherwise resolved (for example, risk avoidance via relaxation of requirements). Typical techniques include prototypes, simulations, benchmarks, mission analyses, key-personnel agreements, design-to-cost approaches, and incremental development.
- Risk monitoring involves tracking the project's progress toward resolving its risk items and taking corrective action where appropriate. Typical techniques include milestone tracking and a top-10 risk-item list that is highlighted at each

weekly, monthly, or milestone project review and followed up appropriately with reassessment of the risk item or corrective action.

In addition, risk management provides an improved way to address and organize the life cycle. Risk-driven approaches, like the spiral model of the software process,[2] avoid many of the difficulties encountered with previous process models like the waterfall model and the evolutionary development model. Such risk-driven approaches also show how and where to incorporate new software technologies like rapid prototyping, fourth-generation languages, and commercial software products into the life cycle.

SIX STEPS

Figure 2 summarized the major steps and techniques involved in software risk management. This overview article covers four significant subsets of risk-management techniques: risk-identification checklists, risk prioritization, risk-management planning, and risk monitoring. Other techniques have been covered elsewhere.[3,4]

Risk-identification checklists. Table 1 shows a top-level risk-identification checklist with the top 10 primary sources of risk on software projects, based on a survey of several experienced project managers. Managers and system engineers can use the checklist on projects to help identify and resolve the most serious risk items on the project. It also provides a corresponding set of risk-management techniques that have been most successful to date in avoiding or resolving the source of risk.

If you focus on item 2 of the top-10 list in Table 1 (unrealistic schedules and budgets), you can then move on to an example of a next-level checklist: the risk-probability table in Table 2 for assessing the probability that a project will overrun its budget. Table 2 is one of several such checklists in an excellent US Air Force handbook[5] on software risk abatement.

Using the checklist, you can rate a project's status for the individual attributes associated with its requirements, personnel, reusable software, tools, and support environment (in Table 2, the environment's availability or the risk that the environment will not be available when needed). These ratings will support a probability-range estimation of whether the project has a relatively low (0.0 to 0.3), medium (0.4 to 0.6), or high (0.7 to 1.0) probability of overrunning its budget.

Most of the critical risk items in the checklist have to do with shortfalls in domain understanding and in properly scoping the job to be done — areas that are generally underemphasized in computer-science literature and education. Recent

TABLE 1.
TOP 10 SOFTWARE RISK ITEMS.

Risk item	Risk-management technique
Personnel shortfalls	Staffing with top talent, job matching, team building, key personnel agreements, cross training.
Unrealistic schedules and budgets	Detailed multisource cost and schedule estimation, design to cost, incremental development, software reuse, requirements scrubbing.
Developing the wrong functions and properties	Organization analysis, mission analysis, operations-concept formulation, user surveys and user participation, prototyping, early users' manuals, off-nominal performance analysis, quality-factor analysis.
Developing the wrong user interface	Prototyping, scenarios, task analysis, user participation.
Gold-plating	Requirements scrubbing, prototyping, cost-benefit analysis, designing to cost.
Continuing stream of requirements changes	High change threshold, information hiding, incremental development (deferring changes to later increments).
Shortfalls in externally furnished components	Benchmarking, inspections, reference checking, compatibility analysis.
Shortfalls in externally performed tasks	Reference checking, preaward audits, award-fee contracts, competitive design or prototyping, team-building.
Real-time performance shortfalls	Simulation, benchmarking, modeling, prototyping, instrumentation, tuning.
Straining computer-science capabilities	Technical analysis, cost-benefit analysis, prototyping, reference checking.

TABLE 2.
QUANTIFICATION OF PROBABILITY AND IMPACT FOR COST FAILURE.

	Probability		
Cost drivers	Improbable (0.0-0.3)	Probable (0.4-0.6)	Frequent (0.7-1.0)
Requirements			
Size	Small, noncomplex, or easily decomposed	Medium to moderate complexity, decomposable	Large, highly complex, or not decomposable
Resource constraints	Little or no hardware-imposed constraints	Some hardware-imposed constraints	Significant hardware-imposed constraints
Application	Nonreal-time, little system interdependency	Embedded, some system interdependencies	Real-time, embedded, strong interdependency
Technology	Mature, existent, in-house experience	Existent, some in-house experience	New or new application, little experience
Requirements stability	Little or no change to established baseline	Some change in baseline expected	Rapidly changing, or no baseline
Personnel			
Availability	In place, little turnover expected	Available, some turnover expected	Not available, high turnover expected
Mix	Good mix of software disciplines	Some disciplines inappropriately represented	Some disciplines not represented
Experience	High experience ratio	Average experience ratio	Low experience ratio
Management environment	Strong personnel management approach	Good personnel management approach	Weak personnel management approach
Reusable software			
Availability	Compatible with need dates	Delivery dates in question	Incompatible with need dates
Modifications	Little or no change	Some change	Extensive changes
Language	Compatible with system and maintenance requirements	Partial compatibility with requirements	Incompatible with system or maintenance requirements
Rights	Compatible with maintenance and competition requirements	Partial compatibility with maintenance, some competition	Incompatible with maintenance concept, noncompetitive
Certification	Verified performance, application compatible	Some application-compatible test data available	Unverified, little test data available
Tools and environment			
Facilities	Little or no modification	Some modifications, existent	Major modifications, nonexistent
Availability	In place, meets need dates	Some compatibility with need dates	Nonexistent, does not meet need dates
Rights	Compatible with maintenance and development plans	Partial compatibility with maintenance and development plans	Incompatible with maintenance and development plans
Configuration management	Fully controlled	Some controls	No controls
	Impact		
	Sufficient financial resources	Some shortage of financial resources, possible overrun	Significant financial shortages, budget overrun likely

initiatives, like the Software Engineering Institute's masters curriculum in software engineering, are providing better coverage in these areas. The SEI is also initiating a major new program in software risk management.

Risk analysis and prioritization. After using all the various risk-identification checklists, plus the other risk-identification techniques in decision-driver analysis, assumption analysis, and decomposition, one very real risk is that the project will identify so many risk items that the project could spend years just investigating them. This is where risk prioritization and its associated risk-analysis activities become essential.

The most effective technique for risk prioritization involves the risk-exposure quantity described earlier. It lets you rank the risk items identified and determine which are most important to address.

One difficulty with the risk-exposure

TABLE 3.
RISK EXPOSURE FACTORS FOR SATELLITE EXPERIMENT SOFTWARE.

Unsatisfactory outcome	Probability of unsatisfactory outcome	Loss caused by unsatisfactory outcome	Risk exposure
A. Software error kills experiment	3-5	10	30-50
B. Software error loses key data	3-5	8	24-40
C. Fault-tolerant features cause unacceptable performance	4-8	7	28-56
D. Monitoring software reports unsafe condition as safe	5	9	45
E. Monitoring software reports safe condition as unsafe	5	3	15
F. Hardware delay causes schedule overrun	6	4	24
G. Data-reduction software errors cause extra work	8	1	8
H. Poor user interface causes inefficient operation	6	5	30
I. Processor memory insufficient	1	7	7
J. Database-management software loses derived data	2	2	4

quantity, as with most other decision-analysis quantities, is the problem of making accurate input estimates of the probability and loss associated with an unsatisfactory outcome. Checklists like that in Table 2 provide some help in assessing the probability of occurrence of a given risk item, but it is clear from Table 2 that its probability ranges do not support precise probability estimation.

Full risk-analysis efforts involving prototyping, benchmarking, and simulation generally provide better probability and loss estimates, but they may be more expensive and time-consuming than the situation warrants. Other techniques, like betting analogies and group-consensus techniques, can improve risk-probability estimation, but for risk prioritization you can often take a simpler course: assessing the risk probabilities and losses on a relative scale of 0 to 10.

Table 3 and Figure 3 illustrate this risk-prioritization process by using some potential risk items from the satellite-experiment project as examples. Table 3 summarizes several unsatisfactory outcomes with their corresponding ratings for P(UO), L(UO), and their resulting risk-exposure estimates. Figure 3 plots each unsatisfactory outcome with respect to a set of constant risk-exposure contours.

Three key points emerge from Table 3 and Figure 3:

♦ Projects often focus on factors having either a high P(UO) or a high L(UO), but these may not be the key factors with a high risk-exposure combination. One of the highest P(UO)s comes from item G (data-reduction errors), but the fact that these errors are recoverable and not mission-critical leads to a low loss factor and a resulting low RE of 7. Similarly, item I (insufficient memory) has a high potential loss, but its low probability leads to a low RE of 7. On the other hand, a relatively low-profile item like item H (user-interface shortfalls) becomes a relatively high-priority risk item because its combination of moderately high probability and loss factors yield a RE of 30.

♦ The RE quantities also provide a basis for prioritizing verification and vali-

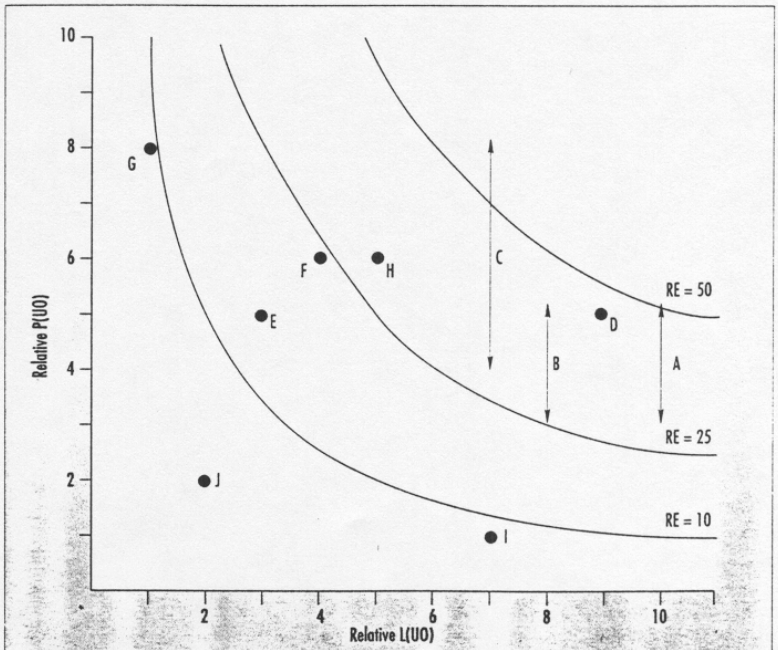


FIGURE 3. RISK-EXPOSURE FACTORS AND CONTOURS FOR THE SATELLITE-EXPERIMENT SOFTWARE. RE IS THE RISK EXPOSURE, P(UO) THE PROBABILITY OF AN UNSATISFACTORY OUTCOME, AND L(UO) THE LOSS ASSOCIATED WITH THAT UNSATISFACTORY OUTCOME. THE GRAPH POINTS MAP THE ITEMS FROM TABLE 3 WHOSE RISK EXPOSURE ARE BEING ASSESSED.

1. Objectives (the "why)
   - Determine, reduce level of risk of the software fault-tolerance features causing unacceptable performance.
   - Create a description of and a development plan for a set of low-risk fault-tolerance features.
2. Deliverables and milestones (the "what" and "when").
   - By Week 3.
     1. Evaluation of fault-tolerance options
     2. Assessment of reusable components
     3. Draft workload characterization
     4. Evaluation plan for prototype exercise
     5. Description of prototype
   - By Week 7.
     6. Operational prototype with key fault-tolerance features.
     7. Workload simulation
     8. Instrumentation and data reduction capabilities.
     9. Draft description, plan for fault-tolerance features.
   - By Week 10
     10. Evaluation and iteration of prototype
     11. Revised description, plan for fault-tolerance features
3. Responsibilities (the "who" and "where")
   - System engineer: G.Smith
     Tasks 1, 3, 4, 9, 11. Support of tasks 5, 10
   - Lead programmer: C.Lee
     Tasks 5, 6, 7, 10. Support of tasks 1, 3
   - Programmer: J.Wilson
     Tasks 2, 8. Support of tasks 5, 6, 7, 10
4. Approach (the "how")
   - Design-to-schedule prototyping effort
   - Driven by hypotheses about fault-tolerance-performance effects
   - Use real-time operating system, add prototype fault-tolerance features
   - Evaluate performance with respect to representative workload
   - Refine prototype based on results observed
5. Resources (the "how much")
   - $60K — full-time system engineer, lead programmer, programmer (10 weeks) * (3 staff) * $2k/staff-week)
   - $0 — three dedicated workstations (from project pool)
   - $0 — two target processors (from project pool)
   - $0 — one test coprocessor (from project pool)
   - $10K — contingencies
   - $70K — total

FIGURE 4. RISK-MANAGEMENT PLAN FOR FAULT-TOLERANCE PROTOTYPING.

dation and related test activities by giving each error class a significance weight. Frequently, all errors are treated with equal weight, putting too much testing effort into finding relatively trivial errors.

- There is often a good deal of uncertainty in estimating the probability or loss associated with an unsatisfactory outcome. (The assessments are frequently subjective and are often the product of surveying several domain experts.) The amount of uncertainty is itself a major source of risk, which needs to be reduced as early as possible. The primary example in Table 3 and Figure 3 is the uncertainty in item C about whether the fault-tolerance features are going to cause an unacceptable degradation in real-time performance. If P(UO) is rated at 4, this item has only a moderate RE of 28, but if P(UO) is 8, the RE has a top-priority rating of 56.

One of the best ways to reduce this source of risk is to buy information about the actual situation. For the issue of fault tolerance versus performance, a good way to buy information is to invest in a prototype, to better understand the performance effects of the various fault-tolerance features.

Risk-management planning. Once you determine a project's major risk items and their relative priorities, you need to establish a set of risk-control functions to bring the risk items under control. The first step in this process is to develop a set of risk-management plans that lay out the activities necessary to bring the risk items under control.

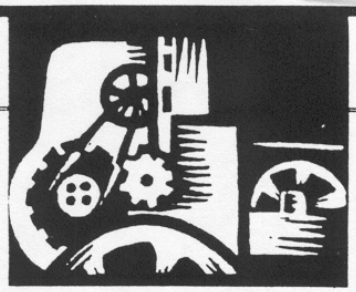
One aid in doing this is the top-10 checklist in Figure 3 that identifies the most successful risk-management techniques for the most common risk items. As an example, item 9 (real-time performance shortfalls) in Table 1 covers the uncertainty in performance effect of the fault-tolerance features. The corresponding risk-management techniques include simulation, benchmarking, modeling, prototyping, instrumentation, and tuning. Assume, for example, that a prototype of representative safety features is the most cost-effective way to determine and reduce their effects on system performance.

The next step in risk-management planning is to develop risk-management plans for each risk item. Figure 4 shows the plan for prototyping the fault-tolerance features and determining their effects on performance. The plan is organized around a standard format for software plans, oriented around answering the standard questions of why, what, when, who, where, how, and how much. This plan organization lets the plans be concise (fitting on one page), action-oriented, easy to understand, and easy to monitor.

The final step in risk-management planning is to integrate the risk-management plans for each risk item with each other and with the overall project plan. Each of the other high-priority or uncertain risk items will have a risk-management plan; it may turn out, for example, that the fault-tolerance features prototyped for this risk item could also be useful as part of the strategy to reduce the uncertainty in items A and B (software errors killing the experiment and losing experiment-critical data). Also, for the overall project plan, the need for a 10-week prototype-development and -exercise period must be factored into the overall schedule, to keep the overall schedule realistic.

Risk resolution and monitoring. Once you have established a good set of risk-management plans, the risk-resolution process consists of implementing whatever prototypes, simulations, benchmarks, surveys, or other risk-reduction techniques are called for in the plans. Risk monitoring ensures that this is a closed-loop process by tracking risk-reduction progress and applying whatever corrective action is necessary to keep the risk-resolution process on track.

Risk management provides managers with a very effective technique for keeping on top of projects under their control: *Project top-10 risk-item tracking*. This technique concentrates management atten-



tion on the high-risk, high-leverage, critical success factors rather than swamping management reviews with lots of low-priority detail. As a manager, I have found that this type of risk-item-oriented review saves a lot of time, reduces management surprises, and gets you focused on the high-leverage issues where you can make a difference as a manager.

Top-10 risk-item tracking involves the following steps:

- Ranking the project's most significant risk items.
- Establishing a regular schedule for higher management reviews of the project's progress. The review should be chaired by the equivalent of the project manager's boss. For large projects (more than 20 people), the reviews should be held monthly. In the project itself, the project manager would review them more frequently.
- Beginning each project-review meeting with a summary of progress on the top 10 risk items. (The number could be seven or 12 without loss of intent.) The summary should include each risk item's current top-10 ranking, its rank at the previous review, how often it has been on the top-10 list, and a summary of progress in resolving the risk item since the previous review.
- Focusing the project-review meeting on dealing with any problems in resolving the risk items.

Table 4 shows how a top-10 list could have worked for the satellite-experiment project, as of month 3 of the project. The project's top risk item in month 3 is a critical staffing problem. Highlighting it in the monthly review meeting would stimulate a discussion by the project team and the boss of the staffing options: Make the unavailable key person available, reshuffle project personnel, or look for new people within or outside the organization. This should result in an assignment of action items to follow through on the options chosen, including possible actions by the project manager's boss.

The number 2 risk item in Table 4, target hardware delivery delays, is also one for which the project manager's boss may be able to expedite a solution — by cutting through corporate-procurement red tape, for example, or by escalating vendor-delay issues with the vendor's higher management.

As Table 4 shows, some risk items are moving down in priority or going off the list, while others are escalating or coming onto the list. The ones moving down the list — like the design-verification and -validation staffing, fault-tolerance prototyping, and user-interface prototyping — still need to be monitored but frequently do not need special management action. The ones moving up or onto the list — like the data-bus design changes and the testbed-interface definitions — are generally the ones needing higher management attention to help get them

TABLE 4.
PROJECT TOP-10 RISK ITEM LIST FOR SATELLITE EXPERIMENT SOFTWARE.

Risk item	Monthly ranking			Risk-resolution progress
	This	Last	No. of months	
Replacing sensor-control software developer	1	4	2	Top replacement candidate unavailable
Target hardware delivery delays	2	5	2	Procurement procedural delays
Sensor data formats undefined	3	3	3	Action items to software, sensor teams; due next month
Staffing of design V&V team	4	2	3	Key reviewers committed; need fault-tolerance reviewer
Software fault-tolerance may compromise performance	5	1	3	Fault-tolerance prototype successful
Accommodate changes in data bus design	6	—	1	Meeting scheduled with data-bus designers
Test-bed interface definitions	7	8	3	Some delays in action items; review meeting scheduled
User interface uncertainties	8	6	3	User interface prototype successful
TBDs in experiment operational concept	—	7	3	TBDs resolved
Uncertainties in reusable monitoring software	—	9	3	Required design changes small, successfully made

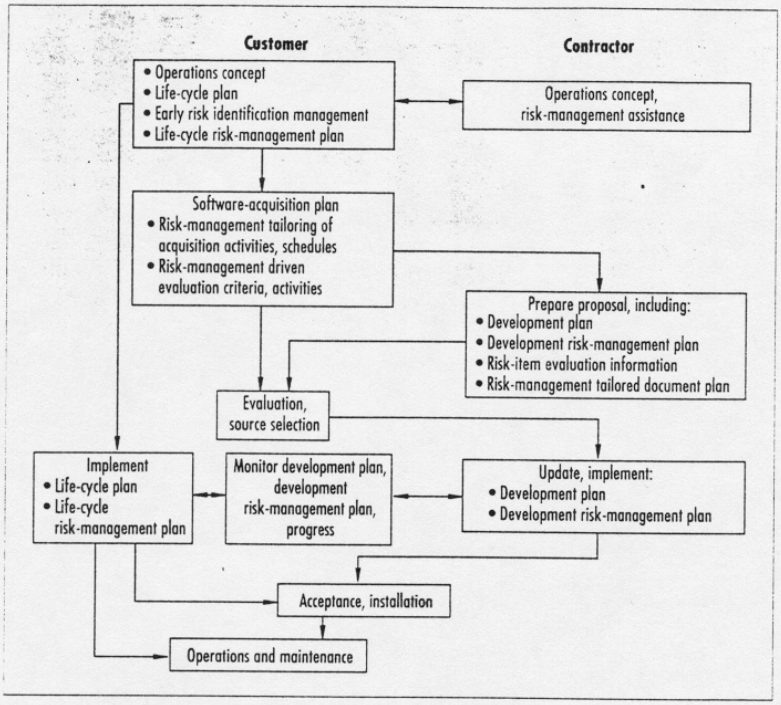


FIGURE 5. FRAMEWORK FOR LIFE-CYCLE RISK MANAGEMENT.

resolved quickly.

As this example shows, the top-10 risk-item list is a very effective way to focus higher management attention onto the project's critical success factors. It also uses management's time very efficiently, unlike typical monthly reviews, which spend most of their time on things the higher manager can't do anything about. Also, if the higher manager surfaces an additional concern, it is easy to add it to the top-10 risk item list to be highlighted in future reviews.

IMPLEMENTING RISK MANAGEMENT

Implementing risk management involves inserting the risk-management principles and practices into your existing life-cycle management practices. Full implementation of risk management involves the use of risk-driven software-process models like the spiral model, where risk considerations determine the overall sequence of life-cycle activities, the use of prototypes and other risk-resolution techniques, and the degree of detail of plans and specifications. However, the best implementation strategy is an incremental one, which lets an organization's culture adjust gradually to risk-oriented management practices and risk-driven process models.

A good way to begin is to establish a top-10 risk-item tracking process. It is easy and inexpensive to implement, provides early improvements, and begins establishing a familiarity with the other risk-management principles and practices. Another good way to gain familiarity is via books like my recent tutorial on risk management,[3] which contains the Air Force risk-abatement pamphlet[5] and other useful articles, and Robert Charette's recent good book on risk management.[4]

An effective next step is to identify an appropriate initial project in which to implement a top-level life-cycle risk-management plan. Once the organization has accumulated some risk-management experience on this initial project, successive steps can deepen the sophistication of the risk-management techniques and broaden their application to wider classes of projects.

Figure 5 provides a scheme for implementing a top-level life-cycle risk-management plan. It is presented in the context of a contractual software acquisition, but you can tailor it to the needs of an internal development organization as well.

You can organize the life-cycle risk-management plan as an elaboration of the "why, what, when, who, where, how, how much" framework of Figure 4. While this plan is primarily the customer's responsibility, it is very useful to involve the developer community in its preparation as well.

Such a plan addresses not only the development risks that have been the prime topic of this article but also operations and maintenance risks. These include such items as staffing and training of maintenance personnel, discontinuities in the switch from the old to the new system, undefined responsibilities for operations and maintenance facilities and functions, and insufficient budget for planned life-cycle improvements or for corrective, adaptive, and perfective maintenance.

Figure 5 also shows the importance of proposed developer risk-management plans in competitive source evaluation and selection. Emphasizing the realism and effectiveness of a bidder's risk-management plan increases the probability that the customer will select a bidder that clearly understands the project's critical success factors and that has established a development approach that satisfactorily addresses them. (If the developer is a non-competitive internal organization, it is equally important for the internal customer to require and review a developer risk-management plan.)

The most important thing for a project to do is to get focused on its critical success factors.

For various reasons, including the influence of previous document-driven management guidelines, projects get focused on activities that are not critical for their success. These frequently include writing boilerplate documents, exploring intriguing but peripheral technical issues, playing politics, and trying to sell the "ultimate" system.

In the process, critical success factors get neglected, the project fails, and nobody wins.

The key contribution of software risk management is to create this focus on critical success factors — and to provide the techniques that let the project deal with them. The risk-assessment and risk-control techniques presented here provide the

foundation layer of capabilities needed to implement the risk-oriented approach.

However, risk management is not a cookbook approach. To handle all the complex people-oriented and technology-driven success factors in projects, a great measure of human judgement is required.

Good people, with good skills and good judgment, are what make projects work. Risk management can provide you with some of the skills, an emphasis on getting good people, and a good conceptual framework for sharpening your judgement. I hope you can find these useful on your next project. ♦

REFERENCES

1. J. Rothfeder, "It's Late, Costly, and Incompetent — But Try Firing a Computer System," *Business Week*, Nov. 7, 1988, pp. 164-165.
2. B.W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, May 1988, pp. 61-72.
3. B.W. Boehm, *Software Risk Management*, CS Press, Los Alamitos, Calif., 1989.
4. R.N. Charette, *Software Engineering Risk Analysis and Management*, McGraw-Hill, New York, 1989.
5. "Software Risk Abatement," AFSC/AFLC pamphlet 800-45, US Air Force Systems Command, Andrews AFB, Md., 1988.

Barry W. Boehm is director of the Defense Advanced Research Project Agency's Information Science and Technology Office, the US government's largest computer/communications research organization. In his previous position as chief scientist for TRW's Defense Systems Group, he was involved in applying risk-management principles to large projects, including the National Aeronautics and Space Administration's space station, the Federal Aviation Administration's Advanced Automation System, and the Defense Dept.'s Strategic Defense Initiative.

Boehm received a BA in mathematics from Harvard University and an MA and PhD in mathematics from UCLA.

Address questions about this article to the author at DARPA ISTO, 1400 Wilson Blvd., Arlington, VA 22209-2308.

Northcon/91

The Premier Electronics Industry Conference of the Northwest
October 1-3, 1991 in Portland, Oregon

Northcon is bringing this comprehensive electronics conference and exhibition to Portland, Oregon, where more than 9,000 electronics engineering professionals - design, test and manufacturing engineers, specifiers, purchasing specialists, engineering management, R&D and corporate personnel, quality specialists and corporate executives - will gather to learn about the latest electronics products and technology.

CALL FOR PAPERS

Individual Papers and Group Sessions are invited.

Papers for presentation in the technical sessions are requested in five areas:

1. *Design/Test/Production/Management*
2. *Computer Hardware/Software Advances*
3. *Research and Development*
4. *Quality and Reliability*
5. *Regulations and Environment*

For a paper to be considered, a 1000-word summary must be submitted that states the objective of the paper, the new contributions, and the conclusion that will be made. Previously published materials are not acceptable.

Abstracts must be mailed or faxed no later than March 15, 1991 to be considered for evaluation.

Please submit abstracts to:
Jon S. Potts
Technical Programs Chair
c/o NC
8110 Airport Boulevard
Los Angeles, CA 90045-3194
(800) 877-2668 or
(213) 215-3976, ext. 251
FAX: (213) 641-5117

Northcon

is a joint venture of:

Portland and Seattle Sections of the Institute of Electrical and Electronics Engineers, IEEE

Electronics Manufacturers Association, EMA

Cascade Chapter of the Electronics Representatives Association, ERA