Software Risk Management:

Principles and Practices

Barry W. Boehm, TRW

1. Introduction

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

Most post-mortems of these software disaster projects 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, which caused them to miss some clear signals of high-risk issues which proved to be the project's downfall later.

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

Current approaches to the software process make it too easy for software 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 if they don't fit other people's ideas, we'll just evolve things until they work." This sort of approach works fine in some well-supported mini-domains such as spreadsheet applications, but in more complex application domains, it most often creates or neglects intrinsic 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 software project managers at work first-hand, and to try to understand and apply the factors that distinguished the more successful project managers from the less successful ones. Some were able to successfully use a waterfall approach, others successfully used an evolutionary development approach, and 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, that's what they were doing. And their projects tended to avoid pitfalls and produce good products.

The recently emerging discipline of software risk management represents 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 software risk items before they become either threats to successful software operation or major sources of software rework.

Risk Management Steps

As seen in Figure 1, the practice of risk management involves two primary steps, Risk Assessment and Risk Control, each with three subsidiary steps. Risk Assessment involves risk identification, risk analysis, and risk prioritization. Risk Control involves risk management planning, risk resolution, and risk monitoring.

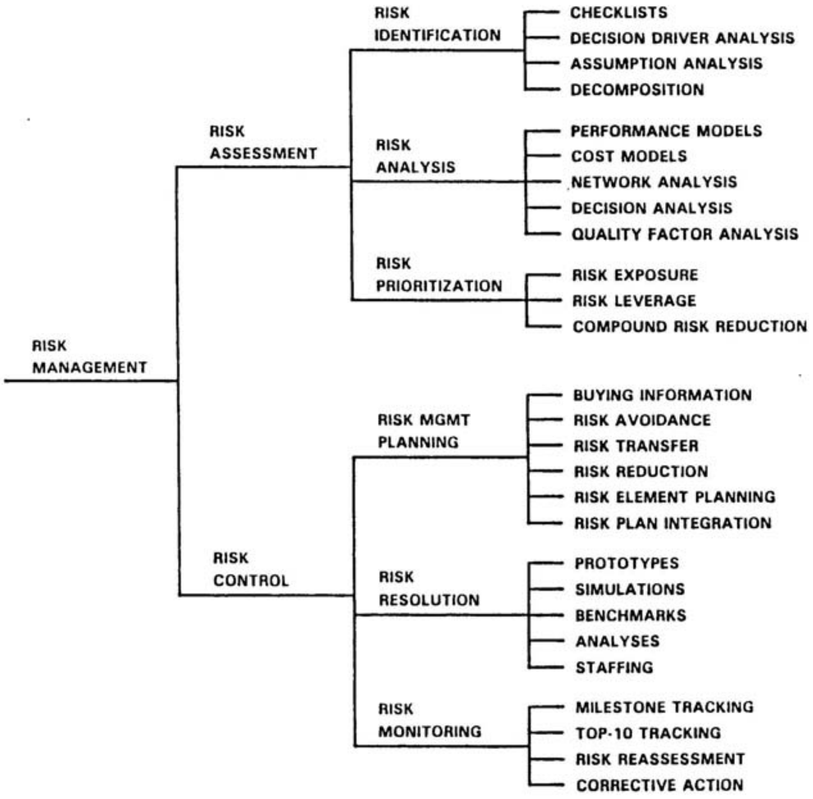
Risk Identification produces lists of the project-specific risk items likely to compromise a project's satisfactory outcome. Typical risk identification techniques include checklists, decomposition, comparison with experience, and examination of decision drivers.

Risk Analysis produces assessments of the loss-probability and loss-magnitude associated with each of the identified risk items, and assessments of compound risks involved in risk-item interactions. Typical techniques include network analysis, decision trees, cost models, performance models, and statistical decision analysis.

Risk Prioritization produces a prioritized ordering of the risk items identified and analyzed. Typical techniques include risk reduction leverage analysis, particularly involving cost-benefit analysis, and Delphi or group-consensus techniques.

Risk Management Planning produces plans for addressing each risk item (e.g., via risk avoidance, risk transfer, risk reduction, or buying information), including the coordination of the individual risk-item plans with each other and with the overall project plan. Typical techniques include checklists of risk

Figure 1. Software Risk Management Steps



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 (e.g., 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 towards resolving its risk items and taking corrective action where appropriate. Typical techniques include risk management plan milestone tracking and a Top Ten Risk Item list which is highlighted at each weekly, monthly, or milestone project review.

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

The next section of this article provides definitions of the basic software risk management terms and concepts. The following section describes and illustrates some of the primary software risk management principles and practices identified in Figure 1. A final section provides an overview of the use of risk management throughout the software life cycle, and presents a overall summary of the article's key points. Next, the companion article to this one [3] will discuss the application of the risk management principles and practices to a large software-intensive application: the FAA Advanced Automation System.

2. Risk Management Definitions and Basic Concepts

Definitions

Webster defines "risk" as "the possibility of loss or injury." This definition can be translated into the fundamental concept of risk management: the concept of Risk Exposure (RE), sometimes also called risk impact. Risk Exposure is defined by the relationship.

$$\mathrm{RE} = \mathrm{Prob}(\mathrm{UO}) * \mathrm{Loss}(\mathrm{UO})$$

where Prob(UO) is the probability of an unsatisfactory outcome and Loss(UO) is the loss to the parties affected if the outcome is unsatisfactory.

To relate this definition to software project situations, we need a definition of "unsatisfactory outcome." Given that software projects involve several classes of participants (customer, developer, user, maintainer), each with somewhat different but highly important satisfaction criteria, we can see 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 qualitv software is unsatisfactory.

These components of an unsatisfactory outcome provide a top-level checklist for identifying and assessing software risk items. More detailed checklists will be covered under Risk Identification below.

Basic Concepts: Risk Exposure and Decision Trees

One of the fundamental risk analysis paradigms is the decision tree. An example is shown in Figure 2. It illustrates a potentially risky situation involving the software controlling a spacecraft experiment. The software has been under development by the experiment team, who understand their experiment well, but are inexperienced 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 which will wipe out the entire experiment and cause an associated loss L(UO) of the total $20 million investment in the experiment.

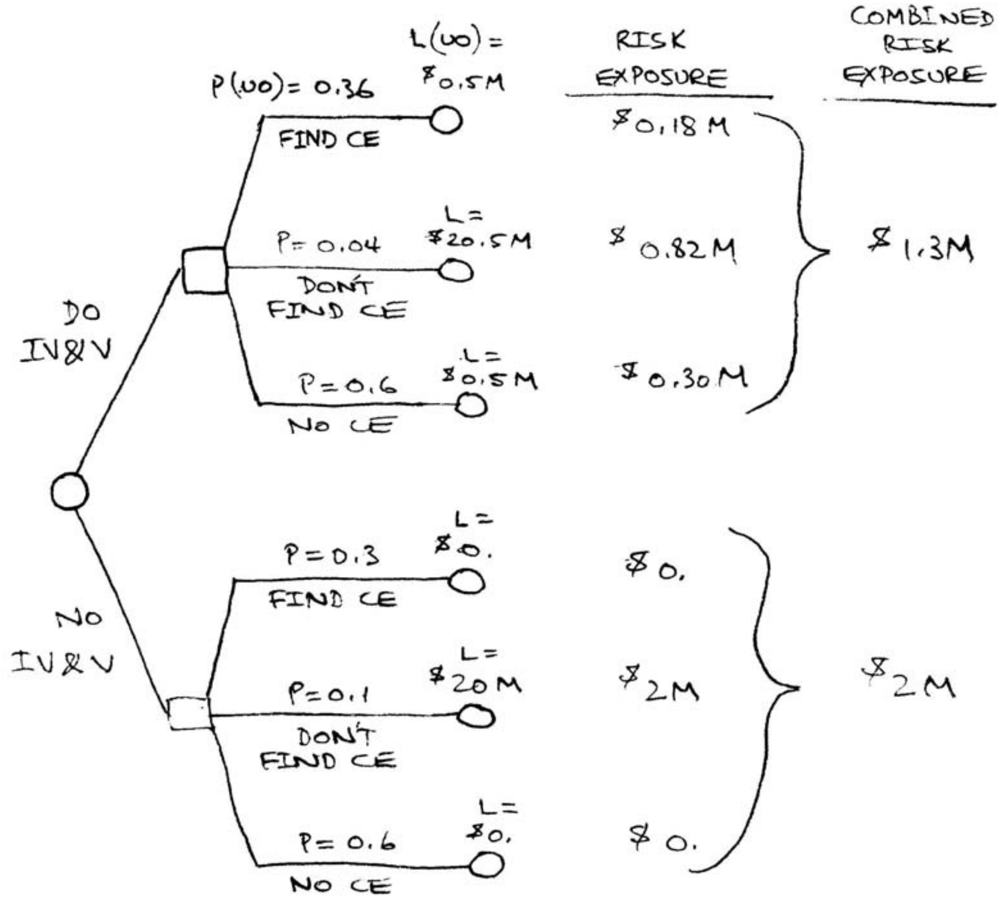
The satellite platform manager identifies two major options for reducing the risk of losing the experiment:

- Convincing and helping the experiment team to apply better software development methods. This incurs no additional cost, and from previous experience the manager estimates that this will reduce the error probability P(U0) to 0.1.

- Hiring a contractor to independently verify and validate (IV&V) the software. This costs an additional $500 thousand; based on the results of similar IV&V efforts, the manager estimates that this will reduce the error probability P(U0) to 0.04.

The decision tree in Figure 2 then shows, for each of the two major decision options, the possible outcomes; 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 $2M. For the IV&V option, the total risk exposure is only $1.3M; thus it represents the more attractive option.

Figure 2. Decision Tree: Spacecraft Experiment Critical Error (CE) Elimination



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 software error were less than $13M, if the experiment team could reduce their critical-software-error probability to less than 0.065, if the IV&V team cost more than $1.2M, if the IV&V team were unable to reduce the probability of critical error to less than 0.075, or if there were various partial combinations of these possibilities.

Even with this sort of sensitivity analysis, there may not be enough information available to quantify the risk exposure parameters well enough to perform a precise analysis. However, the risk exposure framework supports some more approximate but still very useful approaches, such as range estimation and scale-of-10 estimation, which will be discussed in the next section.

3. The Six Steps In Software Risk Management

Figure 1 summarized the major steps and techniques involved in software risk management. Within the confines of this overview article, we do not have space to cover all of the risk management techniques indicated in Figure 1. Thus, we will discuss four of the most significant subsets of risk management techniques: risk identification checklists, risk prioritization, risk management planning, and risk monitoring. More detailed treatment of the other techniques is provided in Reference 4.

Risk Identification Checklists

A top level risk identification checklist is the list shown in Figure 3, showing the top 10 primary sources of risk on software projects, based on a survey of a number of experienced project managers. The checklist can be used by managers and system engineers on an upcoming software project to help identify and resolve the most serious risk items on the project. It also provides a corresponding set of risk management techniques which have been most successful to date in avoiding or resolving the source of risk. An example of the application of this checklist to a large-scale software project is provided in the companion article in this issue [3].

If we focus on item 2 of the top-10 list in Figure 3, "Unrealistic Schedules and Budgets," we can then progress to an example of a next-level checklist: the risk probability table in Figure 4 for assessing the probability that a software project will overrun its budget. Figure 4 is one of several such checklists in an excellent U.S. Air Force handbook (Reference 5) on software risk abatement. Using the checklist, one can rate a software project's status with respect to the individual attributes associated with the project's requirements, personnel, reusable software, tools, and support environment (e.g., in Figure 4, the environment's availability, or the risk that the environment will not be available

Figure 3. *A Top Ten List of Software Risk Items*

RISK ITEM	RISK MANAGEMENT TECHNIQUES
1. Personnel shortfalls	-Staffing with top talent; job matching; teambuilding; morale building; cross-training; prescheduling key people
2. Unrealistic schedules and budgets	-Detailed multisource cost & schedule estimation; design to cost; incremental development; software reuse; requirements scrubbing
3. Developing the wrong software functions	-Organization analysis; mission analysis; ops-concept formulation; user surveys; prototyping; early users' manuals
4. Developing the wrong user interfacing	-Prototyping; scenarios; task analysis
5. Gold plating	-Requirements scrubbing; prototyping; cost-benefit analysis; design to cost
6. Continuing stream of requirements changes	-High change threshold; information hiding; incremental development (defer changes to later increments)
7. Shortfalls in externally furnished components	-Benchmarking; inspections; reference checking; compatibility analysis
8. Shortfalls in externally performed tasks	-Reference checking; pre-award audits; award-fee contacts; competitive design or prototyping; teambuilding
9. Real-time performance shortfalls	-Simulation; benchmarking; modeling; prototyping; intrumentation; tuning
10. Straining computer science capabilities	-Technical analysis; cost-benefit analysis; prototyping; reference checking

Figure 4.

QUANTIFICATION OF PROBABILITY AND IMPACT FOR COST FAILURE

AFSCP 800-45 1987

COST DRIVERS	PROBABILITY		
	IMPROBABLE (0.0 - 0.3)	PROBABLE (0.4 -0.6)	FREQUENT (0.7 - 1.0)
REQUIREMENTS			
SIZE	SMALL, NONCOMPLEX, OR EASILY DECOMPOSED	MEDIUM, 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 INTERDEPENDENCY	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	HIGH TURNOVER, NOT AVAILABLE
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 PDSS REQUIREMENTS	PARTIAL COMPATIBILITY WITH REQUIREMENTS	INCOMPATIBLE WITH SYSTEM OR PDSS REQUIREMENTS
RIGHTS	COMPATIBLE WITH PDSS AND COMPETITION REQUIREMENTS	PARTIAL COMPATIBILITY WITH PDSS, SOME COMPETITION	INCOMPATIBLE WITH PDSS 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 MODIFICATIONS	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 PDSS AND DEVELOPMENT PLANS	PARTIAL COMPATIBILITY WITH PDSS AND DEVELOPMENT PLANS	INCOMPATIBLE WITH PDSS AND DEVELOPMENT PLANS
CONFIGURATION MANAGEMENT	FULLY CONTROLLED	SOME CONTROLS	NO CONTROLS

SUFFICIENT FINANCIAL RESOURCES	SOME SHORTAGE OF FINANCIAL RESOURCES, POSSIBLE OVERRUN	SIGNIFICANT FINANCIAL SHORTAGES, BUDGET OVERRUN LIKELY
IMPACT		

when needed). These ratings will support a probability range estimation of whether the project has a relatively low (0.0-0.3), medium (0.4-0.6), or high (0.7-1.0) probability of overrunning its budget. (The acronym PDSS in Figure 4 stands for Post Development Software Support, often called software maintenance.) References 4 and 5 contain a number of additional useful risk identification checklists.

Another significant point with respect to the top-1 0 list is that most of the critical risk items have to do with shortfalls in domain understanding and in properly scoping the software job to be done -- areas which are generally underemphasized in computer science literature and education.

Risk Analvsis and Prioritization

After using all of the various risk identification checklists, plus the other risk identification techniques involved in decision driver analysis, assumption analysis, and decomposition, one very real risk is that the project will identify so many project 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 which we discussed earlier. It allows us to order the candidate risk items identified and determine which are most important to address.

One difficulty with the RE quantity is the problem of making accurate estimates of the probability and loss associated with an unsatisfactory outcome. Checklists such as Figure 4 provide some help in assessing the probability of occurrence of a given risk item, but it is clear from Figure 4 that the probability ranges in the three columns do not support precise probability estimation. Full risk analysis efforts involving prototyping, benchmarking, simulation, etc., generally provide better probability and loss estimates, but they may be more expensive and time-consuming than the situation warrants. Other techniques, such as betting analogies and group consensus techniques, have been used to improve risk probability estimation, but for the sake of risk prioritization one can often take a simpler course: assessing the risk probabilities and losses on a relative scale of 0 to 10.

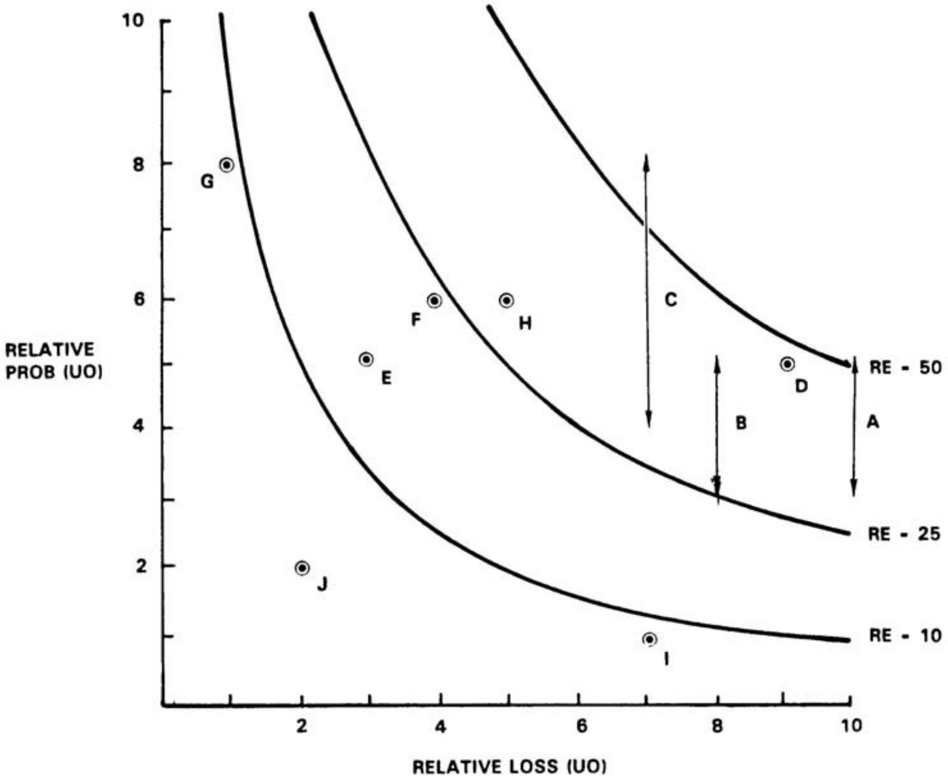
Figures 5 and 6 illustrate this risk prioritization process, by using some potential satellite experiment software project risk items as examples. Figure 5 shows a tabular summary of a number of unsatisfactory outcomes (UO's) with their corresponding ratings for Prob(UO), Loss(UO), and their resulting Risk Exposure estimates. Figure 6 plots each UO with respect to a set of constant risk exposure contours.

Three key points emerge from Figures 5 and 6. First, projects often focus on factors having either a high Prob(U0) or a high Loss(UO), but these may not be the key factors with a high RE combination. One of the highest Prob(U0)'s

Figure 5. Risk Exposure Factors -- Satellite Experiment Software

Unsatisfactory Outcome (UO)	Prob(UO)	Loss(UO)	Risk Exposure
A. S/W error kills experiment	3-5	10	30-50
B. S/W error loses key data	3-5	8	24-40
C. Fault tolerance 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. DBMS software loses derived data	2	2	4

Figure 6. Risk Exposure Factors and Contours:
[redacted] Satellite Experiment Software



comes from item G (data reduction software 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 8. 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 such as 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.

A related second point is that the RE quantities provide a basis for prioritizing V&V and related test activities by giving each class of error a significance weight. Frequently, all errors are treated with equal weight, putting too much effort on relatively trivial errors.

The third key point emerging from Figures 5 and 6 deals with the probability rating ranges given for items A, B, and C. It often occurs that there is 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 different 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 Figures 5 and 6 is the uncertainty in item C about whether the software fault tolerance features are going to cause an unacceptable degradation in real-time performance. If Prob(UO) is rated at 4, this item has only a moderate Risk Exposure of 28; but if Prob(UO) is 8, the RE has a top-priority rating of 56.

One of the best ways of reducing this source of risk due to uncertainty is to buy information about the actual situation. For the issue of fault tolerance features vs. performance, a good way to buy information would be to invest in a prototype, to better understand the performance impact of the various fault tolerance features. We will elaborate on this under Risk Management Planning next.

Risk Management Planning

Once the Risk Assessment activities determine a project's major risk items and their relative priorities, we 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 which 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 which identifies the most successful risk management techniques for the most common risk items on a software project. As an example, the uncertainty in performance impact of the software fault tolerance features is covered under item 9 of Figure 3, Real-Time Performance Shortfalls. The corresponding risk management techniques include simulation, benchmarking, modeling, prototyping, instrumentation, and tuning. Let us assume, for example, that a prototype of representative safety features is the most cost-effective way to determine and reduce their impact on system performance.

The next step in risk management planning is to develop individual Risk Management Plans for each risk item. Figure 7 shows the individual plan for prototyping the fault tolerance features and determining their performance impact. 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 allows the plans to be concise (e.g., fitting on one page), action-oriented, easy to understand, and easy to monitor.

The final step in risk management planning is to integrate the individual 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 or losing experiment-critical data). Also, with respect to the overall project plan, the need for a 10-week prototype development and exercise period must be factored into the overall schedule for the project in order to keep it realistic.

Risk Resolution and Monitoring

Once a good set of risk management plans are established, 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 new keeping technique for on top of projects under their control: Project Top-10 Risk Item Tracking. This technique concentrates management attention on the high-risk, high-leverage critical success factors rather than swamping management reviews with large amounts 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.

Project 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 (over 20 persons), the reviews should be held monthly.

- Beginning each project review meeting with a summary of progress on the top 10 risk items. (The number could be 7 or 12 without loss of

Figure 7. Risk Management Plan: Fault Tolerance Prototyping

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 OS, 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)
$ 0K - 3 dedicated workstations (from project pool)
$ 0K - 2 target processors (from project pool)
$ 0K - 1 test co-processor (from project pool)
$10K - Contingencies
$70K - Total

intent.) The summary should include each risk item's current top-10 ranking, its rank at the previous review, the number of times 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.

Figure 8 shows how a project top-1 0 list could have been working for the spaceborne 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 will stimulate a discussion of the staffing options by the project team and the boss (make the "unavailable" key person available, reshuffle project personnel, look for new people within or outside the organization). This should result in an assignment of action items to follow through on the preferred options chosen, including possible actions by the project manager's boss.

The #2 risk item in Figure 8, target hardware delivery delays, is also one in which the project manager's boss may be able to help 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 seen in Figure 8, some risk items are going down in priority or going off the list, while others are escalating upward or coming onto the list. The ones going down the list, such as the design V&V staffing, fault tolerance prototyping, and user interface prototyping, still need to be monitored but frequently do not need special management action. The ones going up or onto the list, such as the data bus design changes and the testbed interface definitions, are generally the ones needing higher management attention to help in getting them resolved quickly.

As can be seen from this example, the top-10 risk item list is a very effective way to focus higher management attention onto the project's critical success factors. It is also very efficient with respect to management time; the usual monthly review spends most of its 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.

4. Implementing Software Risk Management

Implementing software risk management involves the insertion of the risk management principles and practices into one's existing software life cycle management practices. Full implementation of risk management involves the use of risk-driven software process models such as the spiral model [2], in which 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

Figure 8.

PROJECT TOP 10 RISK ITEM LIST: SATELLITE EXPERIMENT SOFTWARE

	MO. RANKING			
RISK ITEM	THIS	LAST	# MO.	RISK RESOLUTION PROGRESS
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
TESTBED 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

one, which allows an organization's culture to adjust gradually to risk-oriented management practices and risk-driven process models.

A good way to begin is to establish a top-1 0 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 the recent IEEE-CS tutorial volume on Software Risk Management [4], which contains the Air Force Risk Abatement pamphlet [5] and other good articles on the various aspects of implementing software risk management.

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

Figure 9 provides a scheme for implementing a top-level software lifecycle risk management plan. It is presented in the context of a contractual software acquisition, but it can be tailored to an internal software development organization as well.

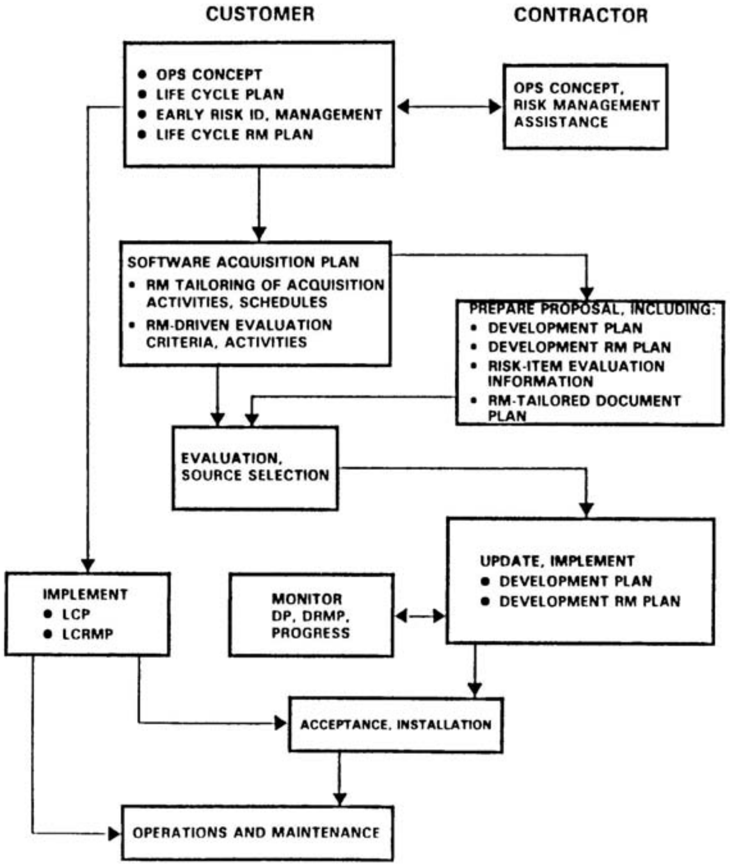
The Software Life Cycle Risk Management Plan can be organized as an elaboration of the "why, what, when, who, where, how, how much" framework given in Figure 7. It is primarily the responsibility of the customer, but it is very useful to involve the developer community in its preparation as well. It addresses not only the development risks which 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 cutover 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 software maintenance.

The other highlight in Figure 9 is 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 who clearly understands the project's critical success factors, and who has established a development approach which satisfactorily addresses them. (If the developer is a noncompetitive internal organization, it is equally important for the internal customer to require and review a developer risk management plan.)

An example of the use of this top-level risk management implementation plan framework is provided in the next article in this issue [3].

Figure 9.

SOFTWARE LIFE CYCLE RISK MANAGEMENT FRAMEWORK


CUSTOMER
CONTRACTOR
• OPS CONCEPT
• LIFE CYCLE PLAN
• EARLY RISK ID, MANAGEMENT
• LIFE CYCLE RM PLAN
OPS CONCEPT, RISK MANAGEMENT ASSISTANCE
SOFTWARE ACQUISITION PLAN
• RM TAILORING OF ACQUISITION ACTIVITIES, SCHEDULES
• RM-DRIVEN EVALUATION CRITERIA, ACTIVITIES
PREPARE PROPOSAL, INCLUDING:
• DEVELOPMENT PLAN
• DEVELOPMENT RM PLAN
• RISK-ITEM EVALUATION INFORMATION
• RM-TAILORED DOCUMENT PLAN
EVALUATION, SOURCE SELECTION
UPDATE, IMPLEMENT
• DEVELOPMENT PLAN
• DEVELOPMENT RM PLAN
IMPLEMENT
• LCP
• LCRMP
MONITOR DP, DRMP, PROGRESS
ACCEPTANCE, INSTALLATION
OPERATIONS AND MAINTENANCE

5. Summary

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

For various reasons, including the influence of previous document-driven software management guidelines, projects get focused on activities which 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 which enable the project to deal with them. The risk assessment and risk control techniques presented in this article, and in more detail in Reference 4, provide the foundation layer of capabilities needed to implement the risk-oriented approach.

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

Good people, with good skills and good judgement, are what make software 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 find these useful on your next software project.

References

1. J. Rothfeder, "It's Late, Costly, and Incompetent - But Try Firing a Computer System," Business Week, November 7, 1988, pp. 164-1 65.

2. B. W. Boehm, "A Spiral Model of Software Development and Enhancement," Computer, May 1988, pp. 61 -72.

3. V.R. Basili, B.W. Boehm, and A.E. Salwin, "Ada Risk Management: A Large-Project Example," Software, (TBD).

4. B.W. Boehm, Tutorial: Software Risk Management, IEEE Computer Society, Catalog No. ISBN-0-81 86-8906-4, 1989.

5. U.S. Air Force Systems Command, "Software Risk Abatement," AFSCIAFLC Pamphlet 800-45, 1988.