Software Risk Management

Barry W. Boehm, University of Southern California
Tom DeMarco, The Atlantic Systems Guild

In mature engineering disciplines, risk management has been *de rigeur* for centuries. When Michelangelo set out to raise the dome of St. Peters in 1547, he was well aware of the potential collapse zones under the staging, the possibility of materials failure, and the human capacity for error. For each of these major risks he prepared a mitigation plan: a fallback, a safety factor, or an alternative.

Today, we routinely practice risk management in our stewardship of the environment, in planning financial strategy, in construction engineering, and in medicine. But how do we apply it to the ultimate risky business, software development?

0740-7459/97/$10.00 © 1997 IEEE

RISK VS. "CAN-DO"

Software development's risky nature is easy enough to acknowledge in the abstract, but sadly, harder to acknowledge in real-world situations. Our culture has evolved such that owning up to risks is often confused with defeatism. Thus, a manager faced with a nearly impossible schedule may deliberately ignore risks to project a confident, "can-do" attitude.

Or is that assessment too severe? After all, *you* wouldn't ignore risks on your project. Or would you? To understand how risks get ignored, we must look beyond the kinds of risk that are subject to easy, obvious mitigation, such as: "If we don't add two folks to the test team right away, we are never going to complete acceptance testing in time for a June 1 delivery." Any manager capable of staying awake during the work day will leap at solving this one; ignoring it means missing a chance to take early and efficient corrective action.

Fatal risks, on the other hand, offer an entirely different challenge. These risks are either beyond effective mitigation or unacceptably costly to mitigate. An example of fatal risk is: "We can now see that the June 1 estimate was hopelessly flawed; we have no chance of making that date, we never had a chance of making it, and most actions we might now take (such as adding more people) can only lengthen the time it will take us to eventually deliver the product." Fatal risks are often ignored by the can-do manager. An attitude that would be obviously stupid in the presence of small risks is somehow considered less stupid for large ones.

Our culture has evolved such that owning up to risks is often confused with defeatism.

FOCUS ON ESSENTIALS

The key concepts of risk management can help software managers assess problem situations and formulate proactive solutions. A good example is the key risk management concept of risk exposure. For each source of problems causing a potential loss to the project, the exposure is defined as the product of the probability of the potential loss, Prob(Loss), multiplied by the size of the loss, Size(Loss):

$$\text{Risk Exposure} = \text{Prob(Loss)} \times \text{Size(Loss)}.$$

A recent gathering of Indian software managers revealed that they perceived personnel turnover as their biggest source of risk. Huge demand for experienced software people—as in the US and elsewhere—confronted the typical software project manager in India with the likely departure of one or more key people, which would potentially result in a substantial and costly disruption of the project's schedule.

The notion of risk exposure helped focus these managers on reducing both Prob(Loss) and Size(Loss). Prob(Loss) can be reduced by assessing, addressing, and monitoring the annual turnover rate. Good strategies for this include empowering performers, teambuilding, establishing significant incentive bonuses for successful project completion, recognizing outstanding efforts, and structuring career paths around an organization's product lines.

Size(Loss) can be reduced significantly too. Software inspections not only find defects, they also spread information on the software product's components across the organization, as do other approaches such as egoless programming and Cleanroom techniques. Modular software architectures and encapsulation confine the effects of personnel turnover to small parts of the system. Software development files and good configuration management make it easier for new replacements to master existing software modules. In combination, a focus on these strategies can make an organization not only more competitive in a high-turnover marketplace, but also a more satisfying place to work.

RISK MANAGEMENT'S DIRTY LITTLE SECRET

Software managers would be more inclined to acknowledge and manage their risks if they were more aware of comparable organizations that had already chosen to move in that direction. Although there is evidence that software risk management is beginning to affect many companies and government agencies, published word of that experience remains minimal. There is a reason why this has been and will probably continue to be true: Doing software risk management makes good sense, but talking about it can expose you to legal liabilities. If a software product fails, the existence of a formal risk plan that acknowledges the possibility of such a failure could complicate and even compromise the producer's legal position. There is reason to suspect that most risk management practitioners have adopted a "don't ask, don't tell" attitude toward publication of their successes.

Given this tendency, we feel particularly lucky to have a lively "Point/Counterpoint" and eight fine articles on risk management in this issue. A ninth risk management article appears in the May issue of our sister publication, *Computer*. Together, these 10 pieces establish a coherent baseline for how risk management is and should be practiced in software organizations.

THE ARTICLES

If you are new to risk management and wondering if you should get involved with it—or at least feel guilty about not yet being involved—start by reading Tim Lister's "Point," which maintains that risk management is project management

for adults. Next, look into Marvin Carr's cautionary "Counterpoint," which asserts that risk management can itself be risky, particularly if it's not done on an institutional basis.

Moving on to the theme articles, "The RAMP Architecture for Assessing and Managing Risk" by Paul Garvey, Douglas Phair, and John Wilson describes a Web-based risk management information system used to capture and apply corporate experience in risk management across a wide variety of projects. Tony Moynihan's "How Experienced Project Managers Assess Risk" summarizes how several managers in medium-sized commercial projects assess the risk implications of a project's situational factors. (He also provides an intriguing partial confirmation of the SEI's risk taxonomy.) Robert Charette, Kevin Adams, and Mary White explore the application of risk management techniques to the world of legacy software in "Managing Risk in Software Maintenance."

Three articles address the interaction of risk management with cost and schedule estimation. Raymond Madachy's "Heuristic Risk Assessment Using Cost Factors" describes an operational expert-system tool that analyzes patterns of cost-driver ratings submitted for a Cocomo cost estimate to determine and rank associated sources of project risk. Kari Känsälä's "Integrating Risk Assessment with Cost Estimation" describes the checklist-based RiskMethod and corresponding RiskTool used to generate both cost and risk estimates, which have been used successfully in several Finnish software companies. In "Estimates, Risk, and Uncertainty," Barbara Kitchenham and Stephen Linkman analyze each of the four major sources of error in using cost and schedule estimation models—measurement error, model error, assumption error, and scope error—and show how each must be considered when managing software cost and schedule uncertainty and risk.

In "Putting Risk Management into Practice," the Software Engineering Institute's Ray C. Williams, Julie A. Walker, and Audrey J. Dorofee use an SEI-designed road map as a guide to discussing effective and ineffective risk management methods in software-intensive Department of Defense programs. Finally, in "Implementing Risk Management" Edmund Conrow and Patty Shishido provide experience-based risk management guidelines from a highly successful distributed command-and-control project that uses several innovative risk management practices. And in the May issue of *Computer*, Art Gemmer's "Risk Management: Moving Beyond Process" examines how one organization approached risk management from the perspective of functional behavior, specifically "how to communicate risk more effectively."

Carr says risk management itself can be risky.

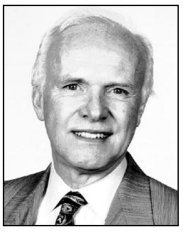
As reported in this special issue, software risk management is quickly becoming a mature discipline. To achieve the promise of fully effective software risk management, the software industry still must address several continuing challenges.

- Achieve commitment of all key stakeholders (developers, customers, users, maintainers, and others) to a risk management approach.
- Establish an evolving knowledge base of risk management experience and expertise, organized for easy and collaborative use by all stakeholders.
- Define and propagate mature guidelines on when and how to avoid, prevent, transfer, or accept and manage risk.
- Develop metrics and tools for reasoning about risk management's return-on-investment issues, including guidelines for deciding how much of a risk reduction activity is enough. Tools that might be used in this way include risk-focused prototyping, specifying, testing, formal verification and validation, configuration management, and quality assurance.

The promising benefits of risk management are evident in the articles we've gathered here. We hope they'll inspire you to think of how you yourself might contribute to further progress in this area. ◆

Barry W. Boehm is the TRW Professor of Software Engineering and Director of the Center for Software Engineering at the University of Southern California. His current research involves the WinWin groupware system for software requirements negotiation, architecture-based models of software quality attributes, and the Cocomo 2.0 cost-estimation model.

Boehm received a BA in mathematics from Harvard University and an MS and a PhD in mathematics from the University of California, Los Angeles. He is a fellow of the IEEE and the AIAA, and a member of the National Academy of Engineering.



Tom DeMarco is a principal of The Atlantic Systems Guild. He is author of *Why Does Software Cost So Much?* (Dorset House, 1995), an editor of *Software State of the Art* (Dorset House, 1990), and co-author (along with Tim Lister) of *Peopleware: Productive Projects and Teams* (Dorset House, 1987). DeMarco is also the winner of the 1986 J.D. Warnier Prize for Lifetime Contribution to the Information Sciences.

Address questions about this article to Boehm at boehm@sunset.usc.edu or to DeMarco at tdemarco@atlsysguild.com.