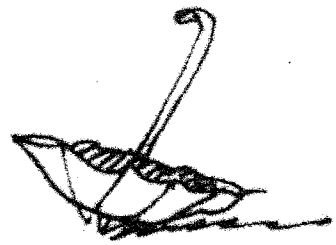


Implementing Risk Management on Software Intensive Projects

EDMUND H. CONROW, *Independent Consultant*
PATRICIA S. SHISHIDO, *TRW Systems Integration Group*

Rising costs, falling performance, and slipping schedules are common problems on large-scale software projects. The authors describe key risk issues and how they were mitigated in one DoD project.

A recent survey by the Standish Group of 365 respondents and 8,380 commercial software-intensive projects indicated that 53 percent of the projects were "challenged": they were over budget, behind schedule, or had fewer features and functions than originally specified, and 31 percent of the projects were canceled.[1] On average, these projects had cost increases of 189 percent and schedule slippage of 222 percent versus the original estimates at the time they were completed or canceled. In addition, the completed projects had an average of only 61 percent of the originally specified features and functions.

Commercial projects do not suffer these problems alone. Software-intensive defense projects also suffer from cost growth, schedule slippage, and performance degradations and are often regarded as high risk. Former US Deputy Assistant Secretary of the Air Force Lloyd K. Mosemann said:

0740-7459/97/$10.00 © 1997 IEEE

TABLE 1
A SUMMARY OF KEY RISK ISSUES

Risk Grouping	Software Risk Issues
Project level	♦ Excessive, immature, unrealistic, or unstable requirements ♦ Lack of user involvement ♦ Underestimation of project complexity or dynamic nature
Project attributes	♦ Performance shortfalls (includes errors and quality) ♦ Unrealistic cost or schedule (estimates and/or allocated amounts)
Management	♦ Ineffective project management (multiple levels possible)
Engineering	♦ Ineffective integration, assembly and test, quality control, specialty engineering, or systems engineering (multiple levels possible) ♦ Unanticipated difficulties associated with the user interface
Work environment	♦ Immature or untried design, process, or technologies selected ♦ Inadequate work plans or configuration control ♦ Inappropriate methods or tool selection or inaccurate metrics ♦ Poor training
Other	♦ Inadequate or excessive documentation or review process ♦ Legal or contractual issues (such as litigation, malpractice, ownership) ♦ Obsolescence (includes excessive schedule length) ♦ Unanticipated difficulties with subcontracted items ♦ Unanticipated maintenance and/or support costs

Software is so vital to military systems that, without it, most could not operate at all. Its importance to overall system performance, and the generally accepted notion that software is always inadequate, makes software the highest risk item and must be steadfastly managed. . . . Failure to identify and address risk has been the downfall of many DoD acquisition programs. The system component with the greatest inherent risk has historically been software."[2]

DEFINING RISK

Risk is the probability (or likelihood) of failing to achieve particular cost, performance, and schedule objectives, and the consequence of failing to achieve those objectives. The word risk is often used incorrectly to represent only the probability term. When used correctly, "risk" represents the combined effect of the probabilty and consequence terms.

Quantitative cost and schedule risk assessment methodologies can ideally estimate risk, since probabilities of occurrence result from the simulation process and the consequence term directly represents either cost (dollars) or schedule (time). Ordinal risk assessment scales can also be used to perform risk analyses. In this case, the word "probability" in the definition of risk given above is replaced by the word "uncertainty." (Uncertainty is especially appropriate for ordinal values, which almost never represent probabilities.) Ordinal assessment scales do not represent risk, the combined effect of uncertainty and consequence, but express the level of uncertainty and/or state of maturity for the "uncertainty" item being evaluated or level of consequence for the impact associated with that item being evaluated and do not require assumptions about probability distributions. True risk values can almost never be mathematically computed from ordinal "uncertainty" and consequence scales and the results from such operations on uncalibrated scales are generally meaningless.[3]

SOURCES OF PROJECT RISK

Numerous studies have identified risk contributors in software-intensive projects. Based upon our examination of many of these studies,[1,2,4-10] we identified 150 candidate risk issues. We then aggregated and summarized these risks, as Table 1 shows.

Additional risk issues. For moderate- and high-complexity development projects, several issues can contribute significantly to increased costs and schedule slippage, such as

♦ using a performance-dominated requirements generation process that begins before you formally start your development process,

♦ starting a project with a budget and schedule that is inadequate for the desired performance level,

♦ using a performance-driven design and development process,

♦ establishing a design that is near the feasible limit of achievable performance (where the magnitudes of the first and second derivatives of cost with respect to performance can be very large),

♦ being overly optimistic in assessing the limits of performance achievable for a given budget and schedule, and

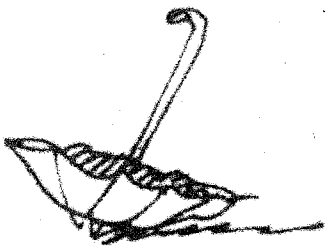
♦ making major project design decisions before the relationship between cost, performance, and schedule is understood.[11]

Each of these items generally contributes to

♦ overoptimism in establishing and estimating adequate project cost and schedule,

♦ underestimation of cost and schedule risk, and

♦ an eventual increase in project cost and schedule during development.[11]



High-risk designs can occur when the buyer and seller focus primarily on increased performance, and thus it comes to dominate the resulting design, even if it increases project risk as well as cost and schedule. The situation is made even worse because the risk level in complex projects is routinely underestimated, often due to human and organizational behavioral issues that are difficult to solve or even identify.[11]

RISK MANAGEMENT IN DEFENSE PROGRAMS

The US Department of Defense has developed a number of policies emphasizing system and software risk management. At the system level, DoD Directive 5000.1[12] states in Section D.1.d:

> *Program managers and other acquisition managers shall continually assess program risks. Risks must be well understood, and risk management approaches developed, before decision authorities can authorize a program to proceed into the next phase of the acquisition process...*

At the software level, *MIL-STD-498*[13] (now becoming *EIA/IEEE J-STD-016*) states in Section 5.19.1:

> *The developer shall perform risk management throughout the software development process. The developer shall identify, analyze, and prioritize the areas of the software development project that involve potential technical, cost, or schedule risks; develop strategies for managing those risks; record the risks and strategies in the software development plan; and implement the strategies in accordance with the plan.*

However, these policies have not been sufficient to achieve successful system and software risk management to date.

Common deficiencies. Four risk management deficiencies have been observed in several DoD development programs.[14] These deficiencies are also often found in civilian development projects.

First, both the buyer (such as the government) and seller (contractor) often have risk management processes that are weakly structured or "ad hoc." There may be no clearly delineated mechanism in place for managing program risk (such as organizational responsibilities, analyses, and products), or if a risk management process exists, it may be weakly implemented or exist on paper only. Even though risk management is required of all major defense development programs, no single set of guidelines exists as to how the risk management process should be implemented. And, although "ad hoc" risk management implementation may be tolerable for projects with relatively low to moderate complexity or technology, it can cause turmoil in projects that push the state of the art.

Second, risk assessment is often too subjective and not adequately documented.

- The prescribed risk assessment categories can be overly broad (such as management, technical). This makes it difficult to evaluate results and implement a viable, measurable risk handling strategy.
- A weak risk assessment methodology can introduce considerable doubt as to the accuracy and value of the results.
- Ordinal risk assessment scales are often incorrectly applied. Mathematical operations cannot be applied to scores obtained from uncalibrated ordinal risk assessment scales. Risk values generated by mathematical operations on uncalibrated ordinal scales will almost always be meaningless and may hide true risk issues.[3]
- The risks may be assigned to broad categories (such as low, medium, and high) without sufficient detail to make the nature of the risk comprehensible.
- The buyer and seller may use different, incompatible risk assessment methodologies, which makes comparing results difficult if not impossible.

Third, the risk assessment process generally emphasizes the probability associated with a specific event and gives less attention to its consequence. However, risk is the combination of an event's probability *and* consequence. You thus must analyze and track both factors over time.

Fourth, program risk assessments and

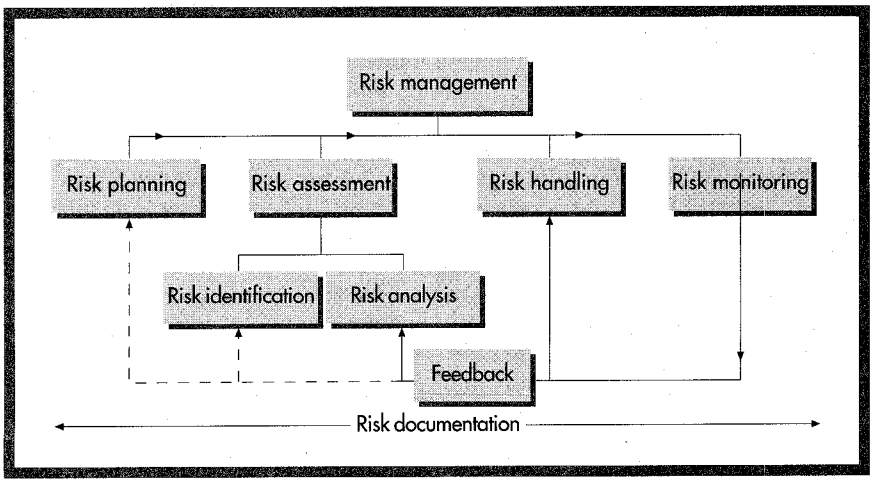
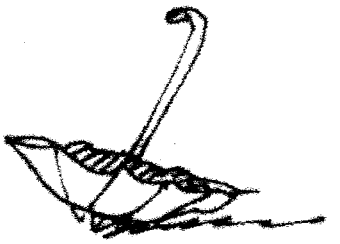


Figure 1. *Good risk management consists of planning, assessment, handling, and monitoring steps coupled in an integrated, closed-loop fashion and performed iteratively throughout the program's life. Figure derived from material courtesy of US DoD.*[15]



risk-handling plans are often unlinked and may be prepared on an as-needed basis, with limited tracking against key program milestones.

Meeting performance requirements was one of our top priorities from the start.

Solid frameworks. To address these deficiencies, the DoD has been evolving risk management frameworks to ensure risk management approaches are complete and consistent. A good recent example is shown in Figure 1,[15] which extends Boehm's risk management framework.[16]

DoD contractors are also elaborating on such frameworks to develop and apply risk-driven process models. The TRW project discussed below successfully applied such a model to complete a software-intensive project with numerous risks within its fixed budget and schedule.

IMPLEMENTING RISK MANAGEMENT

The TRW project is a large software-intensive command and control system developed using the TRW Ada Process Model,[17] an extension of the risk-driven Spiral Model,[5] and a reusable C3 software architecture (network architecture services). The system consists of real-time processing of input messages from various external sources, algorithm processing, data display, and processing output messages to various other external systems with very tight performance and high-reliability requirements. TRW had complete responsibility for system engineering, design, development, and deployment. The operational software consists of over one million lines of Ada source code.

Key challenges. The project presented us with several key challenges.

- Our driving performance requirements were one-second display generations, two-second port-to-port message-processing times, and 50 percent CPU and memory utilization limits.
- Other contractors developed external interfaces concurrently.
- We interacted extensively with users to define algorithms and displays.
- The requirements for the message sets, displays, algorithms, and protocols evolved throughout the life of the project.
- We had a fixed price contract.

To address these challenges, we needed a flexible architecture and design. We also had to uncover potential risk issues early and develop a process that let us make changes easily. To accomplish this, we used "building blocks" of success: measurable metrics built on a foundation of management visibility. The key building blocks consisted of

- a flexible and easily modifiable software architecture,
- a software engineering process that let us make changes throughout the development cycle (TRW Ada Process Model),
- a suitable development environment (toolset), and
- trained personnel.

Our project succeeded because of these building blocks. If we had neglected any one of them, we may have failed to deliver the system on schedule and within budget.

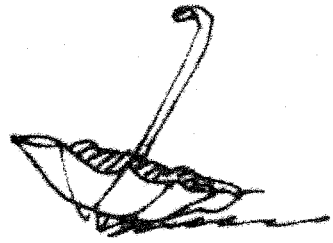
Priorities. Meeting performance requirements was one of our top priorities from the start. The initial project cost and schedule were sufficient to meet the initial performance level required, albeit with identified risk items. When requirements were added and changed, we analyzed the potential impact on cost, performance, and schedule. Because these changes usually increased project complexity, additional risk items were identified and tracked. The TRW Ada Process Model can support requirements changes and new project increments because it has an incremental and evolutionary method of handling new-capability builds. Thus, when new requirements were added after the critical design review, we did not have to restart the whole system development process; we completed the increments under development while planning and designing the new increment so it could be integrated with the previous ones. The priorities associated with meeting cost, performance, schedule requirements, and goals were equally important because we had a fixed price contract, other systems depended on our meeting the delivery schedule, and performance was very important to the users. We completed the project on schedule and within budget. Risk management—along with the proper use of metrics—was an integral part of the management process throughout the project's life cycle.

RISK MANAGEMENT OVERVIEW

Our project had a risk review board led by the project manager. The RRB met monthly during the project's intensive requirements definition, design, and integration phases (during both the concept definition and full scale development contracts). The RRB included representatives from each of the functional and support areas (systems and hardware engineering and test, software, quality assurance, configuration management, and so on) to ensure visibility across the affected areas.

Our risk management process was iterative, with documented feedback to both project management and the customer. This process had four basic steps that map to steps in Figure 1: identification (assessment), assessment (analysis), mitigation planning (handling), and status and control (monitoring).

Risks were documented and included a short description of the risk type (cost, schedule, technical); severity (low, mod-



erate, high), which we determined by qualitatively assessing the potential for occurrence coupled with the potential magnitude of the impact; risk mitigation plan; and status of the risk mitigation activity. Everyone on the project was encouraged to identify risks during any management or technical meeting (such as the software/systems engineering monthly review or in daily engineering activities).

Once a risk item was placed under RRB control, a risk mitigation plan was created and assigned to a responsible person. The RRB then evaluated and approved the plan for implementation, reviewed the status of active risk items/mitigation plans, and modified plans (such as adding new mitigation efforts or modifying existing efforts). Risk items were tracked until the risk was reduced to an acceptable level. These risk items were discussed with the customer through daily interactions and the program management reviews via the Top 10 risks/concerns.

For example, we monitored the estimated system software size versus actuals from the very beginning and found a trend: the software-estimated total size was increasing. Analysis of the software confirmed that the projected growth was real. Our risk mitigation plan resulted in our developing tools that generated Ada source lines of code (commercial tools were not available). This increased productivity, reduced the risk, and helped us meet tight cost and schedule constraints.

KEY RISK ISSUES

The following points show how we dealt with the risk issues identified in Table 1.

♦ *Excessive, immature, unrealistic, or unstable requirements.* Prior to starting this project, this risk was one of the main reasons TRW's large software-intensive projects experienced cost overruns and schedule slips. To address this major risk area, we used the TRW Ada Process Model. With this process model, you specify not just your current project requirements, but plan for their likely directions of growth and change. The model then uses Parnas' information hiding techniques[18] to modularize the software architecture to facilitate anticipated changes. Several other things also helped us contain the potential impact of requirements changes: a well-defined and agreed-upon change control process, metrics to track requirements growth and stability, and allocating capabilities to the various increments.

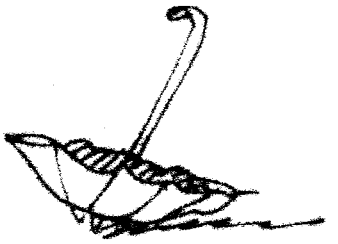
♦ *Lack of user involvement.* Prior to the start of this project, TRW had extensive customer involvement, but little user involvement, and thus large projects with human-machine interfaces often failed to meet user expectations. To address this risk area, the project used extensive prototyping and demonstrated functional capabilities as part of the early reviews (such as design walkthroughs). This prototype evolved into the operational system. Thus, unlike traditional paper design reviews, users could see the displays and interactions to be built into the final system before CDR was completed. To build flexibility into the system, we developed an operational capability to modify display formats. This process model also included a longer design phase to ensure that there was sufficient time to prototype functional capabilities and to get user concurrence. The usual schedule problems that occur during integration did not happen because the actual system integration started during the prototyping activities.

TRW worked with the acquisition customer and end users in working groups to develop the details of displays and algorithms. The representatives had the authority to make technical decisions. These working groups were key. The system was very complex, and early analyses and decisions resolved many users' concerns. Waiting until the traditional end of the development cycle to do the analysis and make decisions to incorporate new enhancements would have led to both cost growth and schedule slippage—not only to the development effort, but to those depending on the system being operational by a certain date.

♦ *Underestimation of project complexity.* Based on previous experience, we knew that changes would occur throughout the development process. We selected the TRW Ada Process Model because its incremental, evolutionary approach would help us address the project's complexity and dynamic nature. The fall of the Berlin Wall, for example, had a major impact on the system's operational concept and requirements.

♦ *Performance shortfalls.* Because we had very tight performance requirements, we addressed this risk during the proposal preparation phase. In the systems engineering area, performance modeling was initiated at the beginning of the project. As the design evolved, the model was updated; when potential problem areas were identified, they were transmitted to the RRB. In addition to the performance modeling activities, the actual software architecture skeleton was built using the network architecture services, and executed with stubs to simulate the processing. Since the software architecture skeleton was built using a code-generation tool, it was easy to change the architecture quickly and with minimal impact to the ongoing design/implementation work. This let us examine different hardware and software design alternatives very early, which minimized the potential cost and schedule impact that might have occurred if a traditional development approach was used. For example, many types of performance problems cannot be identified until the system is integrated, which usually occurs after coding and unit level testing are complete. Again, since we started system integration during the design phase, we

Early analyses and decisions resolved many users' concerns.



could address performance issues early.

♦ *Unrealistic cost or schedule estimates.* We conducted several trade-off studies using the Ada Cocomo cost model to determine a workable combination of functionality, budget, and schedule.

Reviews ensured that we identified problems early and handled them before they affected the project.

♦ *Ineffective project management.* From the start, we had several different types of periodic reviews involving everyone from the hands-on engineer to the project manager. These included reviews of cost, schedule, technology, systems engineering status, software engineering status, and risk review boards. These reviews, together with the use of metrics, ensured that we identified potential problems early and handled them before they affected the project significantly.

♦ *Ineffective integration, assembly and test, quality control, and so on.* We used Ada package specifications during the design phase to consistency-check software interfaces, thus accomplishing significant integration prior to coding rather than waiting until integration testing, which costs more.

♦ *Unanticipated user interface difficulties.* See "Lack of user involvement" above.

♦ *Immature or untried design, processes, or technologies.* The TRW Ada Process Model was a new concept for the customer. To gain customer acceptance, we met several times to discuss it and the implementation approach. To ensure that the process model was working, we monitored metrics throughout the project development phase. For example, we modified the development process slightly when we moved responsibility for system integration from the independent test group to the software development group. This increased efficiency during the prototyping and integration activities and contributed to developers' pride of ownership when their software was integrated into the system.

♦ *Inadequate work plans or configuration control.* Based on TRW's prior experience, we recognized that detailed plans and solid configuration control were very important to project success. We thus developed, maintained, and used detailed work plans and configuration management procedures throughout the development cycle. One of the key configuration control tools was the hierarchical testbed, which automated version control and the software change order tracking process. Quality assurance personnel conducted periodic audits of the various configurations to ensure configuration control was maintained from the beginning of the project to completion of the operational system.

♦ *Inappropriate methods or tool selection or inaccurate metrics.* We evaluated the set of metrics throughout the project. We tailored the set of metrics to insure that it was useful for trend analysis and eliminated extraneous or outdated information to reduce cost and documentation burdens.

♦ *Poor training.* In 1987, the DoD Ada mandate was fairly new and, because Ada was a new language, TRW did not have many Ada developers. Prior to the project contract award, TRW set up Ada language training for more than 100 people. TRW set up additional training for software engineers, including integrators and testers, on the Rational hardware development environment. We also had a training plan that tracked and identified the required and suggested training for each software engineer on the project.

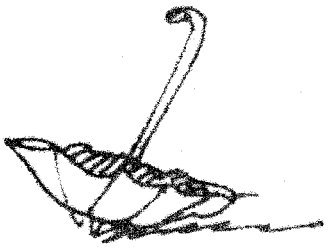
♦ *Inadequate or excessive documentation or review process.* Data item descriptions were tailored prior to the contract award and throughout the various project phases. The minimized redundancy helped reduce the documentation that had to be generated, maintained, and reviewed.

Early in the project, customer team members made numerous comments on the many products they reviewed. These comments were then analyzed, categorized, and prioritized by the commentators. We gave the most attention to the higher priority comments. Meetings (which included commentators) were held to resolve comments and incorporate agreed upon changes. This eliminated several cycles of comments and revisions that usually occur when there is no dialogue between the commentator and document author.

♦ *Legal or contractual issues.* These did not apply on this project.

♦ *Obsolescence.* Available technology changed at an accelerating pace over the course of the project and both hardware and software COTS became obsolete more quickly than we had previously experienced. Project and vendor personnel worked closely together, and the vendors provided early notice when a product was planned for upgrade or obsolescence. When we learned of either a product's revision or its impending obsolescence, we managed it as a risk item. The changes were analyzed to determine if the system would be affected. We then prototyped the revised or replacement product and integrated it with the system. We did this off-line and did not change the software baseline until the product had been thoroughly prototyped and we were confident it could be replaced. This required changes to the application code on numerous occasions, but if and when the new product was placed into the system, it was a planned activity.

♦ *Unanticipated difficulties with subcontracted items.* During the project development phase, we assessed subcontracted items from a particular vendor to be risky because of the vendor's questionable "financial health." We identified the vendor's financial status as a risk item and developed and implemented a risk mitigation plan. This plan included research to determine if there were any other vendors who could



support the project's requirements. When we determined that this was not the case, we initiated discussions with the vendor to start an escrow account for all the engineering information and copies of their software code, in case they could no longer support the project's needs. The vendor is now stronger financially and the escrow account is no longer necessary.

♦ *Unanticipated maintenance and/or support costs.* Maintenance costs grew at a faster pace than we originally planned due to changes in the vendor cost rate structures, product changes, and hardware upgrades. We negotiated changes with the vendors to minimize the impact of these costs and worked out mutually beneficial terms.

The TRW Ada Process Model, which supports constantly changing requirements, has evolved into the TRW Data Technologies Division standard software development process. Instantiations of this process are now being used throughout industry.

We used risk management throughout the life of our project as an integral part of the management process. Risk management is an iterative process. Each project's risks are unique and must be identified, analyzed, mitigated, and tracked. When you encourage people to identify risk items (that is, you don't shoot the messenger), potential risks are often identified early enough to let you act rather than react. Our project also shows that appropriate metrics (which evolve with the project phases) are a necessary tool to help with this early risk identification. A well-defined and disciplined risk management process can increase the level of communication both vertically and horizontally. ◆

REFERENCES

1. The Standish Group International, *Charting the Seas of Information Technology*, Dennis, Mass., 1994.
2. L.K. Mosemann, *Guidelines for Successful Acquisition and Management of Software Intensive Systems*, Dept. of Defense, Version 1.1 – Vol. 1, Feb. 1995.
3. E.H. Conrow, "The Use of Ordinal Scales in Defense Systems Engineering," *Proc. 1995 Acquisition Research Symp.*, Defense Systems Mgt. College, 1995, pp. 455-463.
4. Air Force Systems Command, "Software Risk Management," AFSC Pamphlet 800-45, Sept. 1988.
5. B.W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, May 1988, pp. 61-72.
6. R.N. Charette, *Software Engineering Risk Analysis and Management*, McGraw-Hill, New York, 1989.
7. M. Evans, "Thread of Failure: Project Trends That Impact Success and Productivity," *NetFocus*, No. 203, Software Program Managers Network, Naval Information System Management Center, Mar. 1994.
8. C. Jones, "Assessment and Control of Software Risks," *Proc. 2nd Software Eng. Inst. Conf. Software Risks*, SEI, Carnegie Mellon Univ., Pittsburgh, 1993.
9. C. Jones, "Risks of Software System Failure or Disaster," *American Programmer*, Mar. 1995, pp. 2-9.
10. F.J. Sisti and S. Joseph, "Software Risk Evaluation Method," Tech. Report CMU/SEI-94-TR-19, SEI, Carnegie Mellon Univ., Pittsburgh, 1994.
11. E.H. Conrow, "Some Long-Term Issues and Impediments Affecting Military Systems Acquisition Reform," *Acquisition Review Quarterly*, Summer 1995, pp. 199-212.
12. Dept. of Defense, "Defense Acquisition," DoD Directive 5000.1, Mar. 1996.
13. Dept. of Defense, "Software Development and Documentation," *MIL-Standard-498*, Dec. 1994.
14. E.H. Conrow and M.A. Fredrickson, "Some Considerations for Implementing Risk Management in Defense Programs," *Program Manager*, Defense Systems Mgt. College, Jan.-Feb. 1996, pp. 6-11.
15. Dept. of Defense, *Defense Acquisition Deskbook*, Version 1.3, Dec. 31, 1996.
16. B.W. Boehm, *Software Risk Management*, IEEE Comp. Soc. Press, Los Alamitos, Calif., 1989.
17. W. Royce, "TRW's Ada Process Model for Incremental Development of Large Software Systems," *Proc. ICSE 12*, Mar. 1990, pp. 2-11.
18. D.L. Parnas, "Designing Software for Ease of Extension and Contraction," *IEEE Trans. Software Eng.*, Mar. 1979, pp. 128-137.

Edmund H. Conrow is the founder and owner of Acquisition and Technology Associates. Conrow has more than 20 years' experience applying project management and technical skills to moderate to high complexity programs. He has served a broad range of clients, including: industry, federally funded research centers, national laboratories, and government. His practice is focused on acquisition strategy, engineering design analysis, risk management, and systems engineering. Conrow has developed a variety of risk management approaches and successfully implemented them on numerous programs.

Conrow received a BS and an MS in nuclear engineering from the University of Arizona, a PhD in general engineering from Oklahoma State University, and a PhD in public policy analysis from the RAND Graduate School. He is a senior member of AIAA and a member of the IEEE.

Patricia Shishido is an Integrated Product Team (IPT) lead on TRW's Optima21, an automated framework under development that will utilize and improve existing methods and tools to enable organizations to be established and run more efficiently. Most recently, she was project manager of a major DoD contract where this risk management program was implemented. Her 20 years of experience includes both management and technical responsibilities in systems engineering and software engineering disciplines—including requirements management, design, development, integration, test, and maintenance—on large and complex systems for various government customers.

Shishido holds a BS in mathematics from the University of Redlands, with foreign studies at Kansai University in Osaka Japan.

Address questions about this article to Conrow at (310) 374-7975 or ehc@earthlink.net.