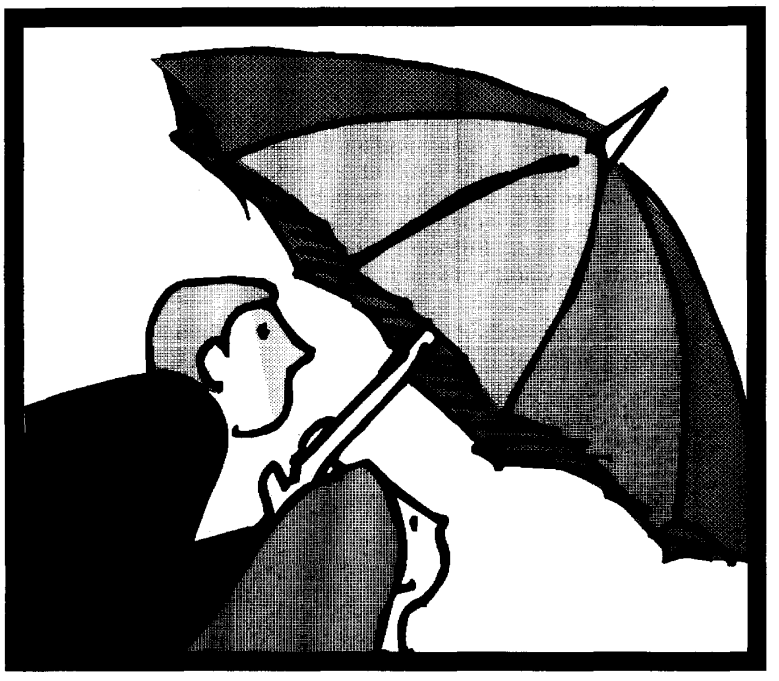
FEATURE

RISK MANAGEMENT FOR SOFTWARE PROJECTS

There is little to instruct software project managers on how to handle risk in a way that ensures the success of contingency planning and avoids crisis. This seven-step procedure describes how to identify risk factors, calculate their probability and effect on a project, and plan for and conduct risk management.



RICHARD FAIRLEY
Software Engineering Management Associates

Many software projects fail to deliver acceptable systems within schedule and budget. Many of these failures might have been avoided had the project team properly assessed and mitigated the risk factors, yet risk management is seldom applied as an explicit project-management activity. One reason risk management is not practiced is that very few guidelines are available that offer a practical, step-by-step approach to managing risk. To address this deficiency, I have created a seven-step process for risk management that can be applied to all types of software projects.

I base the process on several years of work with numerous organizations to identify and overcome risk factors in software projects. My clients and I have used a variety of risk-management techniques within the framework of the process. I describe one set of techniques here, which incorporates regression-based cost modeling, but other techniques, such as decision theory, risk tables, and spiral process models, are equally applicable.[1]

ELEMENTS OF RISK MANAGEMENT

The seven steps of my risk-management process are

1. Identify risk factors. A risk is a potential problem; a problem is a risk that has materialized. Exactly when the transformation takes place is somewhat subjective. A schedule delay

0740-7459/94/$04.00 © 1994 IEEE

of one week might not be cause for concern, but a delay of one month could have serious consequences. The important thing is that all parties who may be affected by a schedule delay agree in advance on the point at which a risk will become a problem. That way, when the risk does become a problem, it is mitigated by the planned corrective actions. In identifying a risk, you must take care to distinguish symptoms from underlying risk factors. A potential schedule delay may in fact be a symptom of difficult technical issues or inadequate resources.

Whether you identify a situation as a risk or an opportunity depends on your point of view. Is the glass half full or half empty? Situations with high potential for failure often have the potential for high payback as well. Risk management is not the same as risk aversion. Competitive pressures and the demands of modern society require that you take risks to be successful.

2. Assess risk probabilities and effects on the project. Because risk implies a potential loss, you must estimate two elements of a risk: the probability that the risk will become a problem and the effect the problem would have on the project's desired outcome. For software projects, the desired outcome is an acceptable product delivered on time and within budget. Factors that influence product acceptability include delivered functionality, performance, resource use, safety, reliability, versatility, ease of learning, ease of use, and ease of modification.

Depending on the situation, failure to meet one or more of these criteria within the constraints of schedule and budget can precipitate a crisis for the developer, the customer, and/or the user community. Thus, the primary goal of risk management is to identify and confront risk factors with enough lead time to avoid a crisis.

The approach I describe here is to assess the probability of a risk by computing probability distributions for

REGRESSION-BASED COST MODELING

You develop a regression-based cost model by collecting data from past projects for relationships of interest (like software size and required effort), deriving a regression equation, and incorporating additional cost factors to explain deviations of actual project costs from the costs predicted by the regression equation.

A commonly used approach to regression-based cost modeling is to derive a linear equation in the log-log domain (log Effort, E, as a linear slope-intercept function of log Size, S) that minimizes the residuals between the equation and the data points for actual projects. Transforming the linear equation, $\log E = \log a + b \bullet \log S$, from the log-log domain to the real domain gives you an exponential relationship of the form $E = a * S^b$. Figure A illustrates this approach, where E is measured in person-months and S is measured in thousands of lines of source code (KLOC).

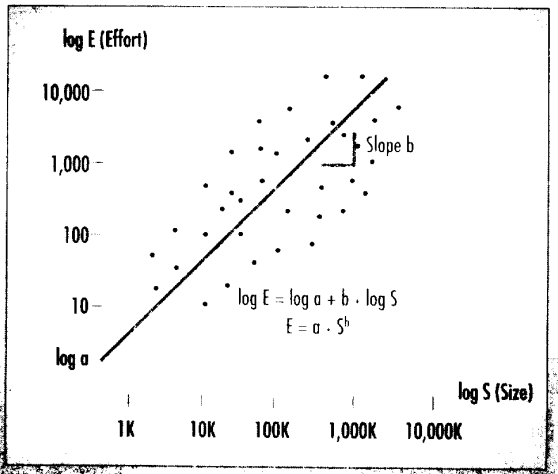


Figure A. Derivation of a regression-based cost model.

As the figure shows, it is not untypical to observe wide scatter in actual project data, which indicates large variations in the effort predicted by the regression equation and the actual effort. Residual error is one measure of the variations. A large residual error means that factors in addition to size exert a strong influence on required effort. If size were a perfect predictor of effort, every data point in Figure A would lie on the line of the equation, and the residual error would be zero.

The next step in regression-based cost modeling is to identify the factors that cause variations between predicted and actual effort. We might, by examining our past projects, determine that 80 percent of the variation in required effort for projects of similar size and type can be explained by variations in stability of the requirements, familiarity of the development team with the application domain, and involvement of users during the development cycle. As illustrated in Table A, you can assign weighting factors to these variables to model their effects.

The regression-based cost model is then of the form:

$$\text{Effort} = (a * \text{Size}^b) * \text{EAF}$$

where EAF is the effort-adjustment factor, which is the product of the effort multiplier values from Table A. According to the table, high requirements volatility,

code size and complexity and use them to determine the effect of limited target memory and execution time on overall project effort. I then use Monte Carlo simulation to compute the distribution of estimated project effort as a function of size, complexity, timing, and memory, using regression-based modeling.

This approach uses estimated effort as the metric to assess the impact of risk factors. Because effort is the primary cost factor for most software projects, you can use it as a measure of overall project cost, especially when using loaded salaries (burdened with facilities, computer time, and management, for example).

3. Develop strategies to mitigate identified risks. In general, a risk becomes a problem when the value of a quantitative metric crosses a predetermined threshold. For that reason, two essential parts of risk management are setting thresholds, beyond which some corrective action is required, and determining ahead of time what that corrective action will be. Without such planning, you quickly realize the truth in the answer to Fred Brooks' rhetorical question, "How does a project get to be a year late?" One day at a time.[2]

Risk mitigation involves two types of strategies. *Action planning* addresses risks that can be mitigated by immediate response. To address the risk of insufficient experience with a new hardware architecture, for example, the action plan could provide for training the development team, hiring experienced personnel, or finding a consultant to work with the project team. Of course, you should not spend more on training or hiring than would be paid back in increased productivity. If you estimate that training and hiring can increase productivity by 10 percent, for example, you should not spend more than 10 percent of the project's personnel budget in this manner.

Contingency planning, on the other hand, addresses risks that require mon-

medium application experience, and low user involvement would result in an EAF of 1.56 (1.2 * 1.0 * 1.3); low requirements volatility, medium application experience, and high user involvement would result in an EAF of 0.64 (0.8 * 1.0 * 0.8). The former situation would require 56 percent more effort than the nominal estimate, while the latter would require 36 percent less effort than the nominal case.

Using effort multipliers to adjust an estimate implies that factors not accounted for in the model do not change from past projects to the one being estimated. For example, the model presented in Figure A and Table A does not incorporate factors such as personnel capabilities or stability of the development environment. If these factors should change, the corresponding impacts (positive or negative) must be incorporated into the estimate for a future project. Failure to do so increases risk.

Examples. Barry Boehm's Cocomo (Constructive Cost Model) is perhaps the best known example of a regression-based cost model. Cocomo is based on data from 63 projects, collected by Boehm during the mid-to-late 1970s. He clustered the data into three groupings, which he called modes. He then derived two linear equations for each mode in the log-log domain; one equation for estimated effort as a function of software size and one for estimated development time as a function of estimated effort.

Boehm and his colleagues identified 15 cost drivers as those factors that contributed most to the observed variations in effort and schedule for software projects of similar size and mode. The ranges of cost-driver values were derived by expert judgment using a Delphi procedure.

TABLE A
EFFORT MULTIPLIERS FOR A SOFTWARE PROJECT

Cost driver	Effort multiplier		
	Low	Medium	High
Requirements volatility	0.8	1.0	1.2
Application experience	1.4	1.0	0.7
User involvement	1.3	1.0	0.8

Boehm illustrated, by example, how to construct a regression-based cost model; hence the name of the model. The model does not work without recalibration to allow for differences in Boehm's environment and the environment of interest, however. When organizations use the equations and tables without doing so, the estimates may be seriously skewed. ***Cocomo equations and tables should not be used as published without recalibrating the model in the local environment.***

Automation concerns. Several tools are available that automate regression-based cost modeling. One of the best tool sets, for versatility and ease of use, is from the Softstar Systems Company of Amherst, New Hampshire. The Softstar tools include a tool (Calico) for entering local project data and deriving regression equations tailored to the local environment, a tool (Dbedit) to edit the effort and schedule distribution tables, cost-driver values, hours per work-month, and other factors, and the estimation tool (CoStar), which uses the outputs from Calico and Dbedit as the basis for estimates.

itoring for some future response should the need arise. To mitigate the risk of late delivery by a hardware vendor, for example, the contingency plan could provide for monitoring the vendor's progress and developing a software emulator for the target machine.

Of course, the risk of late hardware delivery must justify the added cost of preparing the contingency plan, monitoring the situation, and implementing the plan's actions. If the cost is justified, plan preparation and vendor monitoring might be implemented immediately, but the action to develop an emulator might be postponed until the risk of late delivery became a problem (the vendor's schedule slipped beyond a predetermined threshold). This brings up the issue of sufficient lead time. When do you start to develop the emulator? The answer lies in analyzing the probability of late delivery. As that probability increases, the urgency of developing the emulator becomes greater.

4. Monitor risk factors. You must monitor the values of risk metrics, taking care that the metrics data is objective, timely, and accurate. If metrics are based on subjective factors, your project will quickly be reported as 90 percent complete and remain there for many months. You must avoid situations in which the first 90 percent of the project takes the first 90 percent of the schedule, while the remaining 10 percent of the project takes another 90 percent of the schedule.

5. Invoke a contingency plan. A contingency plan is invoked when a quantitative risk indicator crosses a predetermined threshold. You may find it difficult to convince the affected parties that a serious problem has developed, especially in the early stages of a project. A typical response is to plan on catching up during the next reporting period, but most projects never catch up without the explicit, planned corrective actions of a contingency plan. You must also specify the duration of each contingency plan to avoid contingent actions of interminable duration. If the team cannot solve the problem within a specified period (typically one to two weeks), they must invoke a crisis-management plan.

6. Manage the crisis. Despite a team's best efforts, the contingency plan may fail, in which case the project enters crisis mode. There must be some plan for seeing a project through this phase, including allocating sufficient resources and specifying a drop-dead date, at which time management must reevaluate the project for more drastic corrective action (possibly major redirection or cancellation of the project).

7. Recover from a crisis. After a crisis, certain actions are required, such as rewarding personnel who have worked in burnout mode for an extended period and reevaluating cost and schedule in light of the drain on resources from managing the crisis.

I illustrate these seven steps for a project to implement a telecommunications protocol. The project, which is actually a composite of several real projects, gave me the opportunity to explore key risk-management issues, such as the likelihood that an undesired situation might occur, the resulting effect of the risk situation, the cost of mitigating the risk, the degree of urgency in mitigation, and the lead time required to avoid a crisis.

CASE STUDY

The project's goal was to implement a telecommunications protocol for a network gateway using a 10-MHz microprocessor with a 256-Kbyte memory. The project had several constraints that challenged risk management. The project team could not enlarge the memory because the processor was provided by the customer and its use was mandatory. The maximum execution time for message processing was 10 ms.

Risk identification. I used a regression-based cost model to identify and assess the impact of risk factors on estimated project effort. The box on pp. 58-59 describes regression-based cost modeling in more detail, as well as some tools for automating it. For the telecom project, I used a regression-based cost model for real-time telecommunications systems on microprocessors, which I had developed for the client, using historical data from similar projects.

The regression equation I derived to relate effort to product size is

$$\text{Effort} = 3.6 \cdot (\text{Size})^{1.25} \cdot \text{EAF}$$

where EAF is the effort-adjustment factor. EAF is the product of 15 cost factors taken from Barry Boehm's Cocomo model:[3] Required software reliability (Rely), ratio of database size to source-code size (Data), software complexity (Cplx), execution time constraint on the target machine (Time), memory constraint on the target machine (Stor), volatility of the development machine and software (Virt), response time of the development environment (Turn), analyst capability (Acap), applications experience for the development team (Aexp), programmer capability (Pcap), team experience on the development environment (Vexp), team experience with the programming language (Lexp), use of modern programming practices (Modp), use of software tools (Tool), and required development schedule (Sced).

Using these cost drivers as a checklist for the telecom project, I identified five risk factors and added one (Size):

- *Cplx.* Effect of algorithmic complexity
- *Time.* 10-ms timing constraint
- *Stor.* 256K memory of the target processor
- *Vexp.* Lack of experience with the target processor
- *Tool.* Lack of adequate software tools for the target processor

♦ *Size.* Uncertainty in estimated code size.

These six factors are interrelated: If the algorithms are complex, code size is likely to increase; if size increases, more memory and execution time will be required. With more experience on the target processor architecture and with better software tools, the team might better control the code size, execution time, and memory requirements.

Probability and effects assessment. According to evidence from similar projects and some analysis, I estimated that the size of the telecom project's code would be no less than 9 KLOC and no more than 15 KLOC, with the most likely size being approximately 10 KLOC, as Figure 1a shows. Figure 1b is the probability-density function for the probable effect of algorithmic complexity (Cplx) on project effort. As the figure shows, I estimated the most likely impact to be 1.3, with a normal distribution of 1.0 to 1.6. The function for Cplx models the impact that uncertainty in target-machine experience (Vexp) and lack of tools (Tool) will have on the ability to control the complexity of the program that implements the communication algorithms. I used these probability-density functions to derive a distribution of probable project effort, as the box on p. 62 describes.

Thus, the risk factors to be modeled are software size, algorithmic complexity, and the memory and execution-time constraints of the target machine. To assess the effect of uncertainty in size, complexity, execution time, and the memory constraint on the required effort, I constructed a probabilistic cost model and used Monte Carlo simulation. The simulation model is of the form

$$\text{Effort} = 3.6 * (\text{Size})^{1.25} * \text{EAF} \quad (1)$$

where EAF is the product of Stor, Time, and Cplx, and where Size and Cplx are modeled by the probability distributions in Figure 1. Stor and Time are dependent on Size.

I determined values for Stor by first randomly selecting a value from the inverse probability distribution for Size. I then used a code-expansion factor of 16 (based on a ratio of 1 to 4 for source-to-object instructions and 1 to 4 for object instructions to object bytes), multiplied by Size, and divided by 256K (the memory size) to get the percentage of memory used. That is,

TABLE 1
EFFECTS OF LIMITED MEMORY AND TIME ON PROJECT EFFORT

Memory used	Stor	Time used	Time
Less than 50%	1.00	Less than 50%	1.00
70%	1.06	70%	1.11
85%	1.21	85%	1.30
95%	1.56	95%	1.66

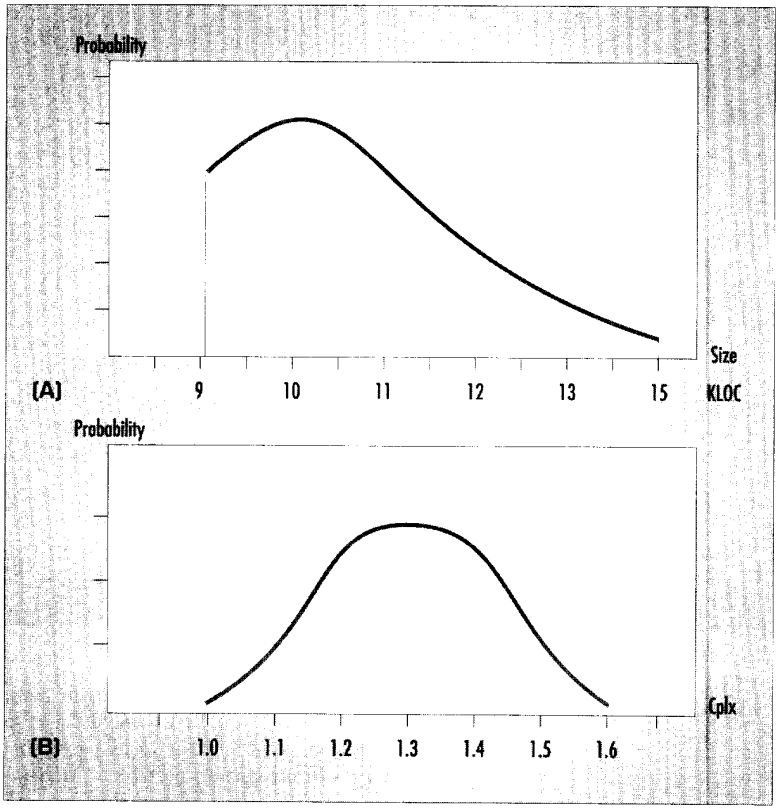


Figure 1. *Two probability distribution curves for the telecom project.* ***(A)*** *Size distribution and* ***(B)*** *effect of algorithmic complexity (Cplx) on project effort.*

PROBABILITY-DENSITY AND DISTRIBUTION FUNCTIONS

Probability-density functions [$p(x)$] are the continuous counterparts of discrete probability histograms (the relative number of times you can expect event x to occur). A probability-distribution function is the integral of $p(x)$. Integrating $p(x)$ from negative infinity to Y is the probability that x will be less than or equal to Y:

$$P(x \leq Y) = \int p(x)\, dx$$

where $p(x)$ is the lognormal distribution function in Figure 1 in the main text, for example. The integral $P(Y \leq x \leq Z)$ is the continuous counterpart of summing the values of a discrete probability histogram from Y to Z:

$$P(Y \leq x \leq Z) = \int p(x)\, dx$$

This integral is the probability that x will be in the range Y to Z; for example, the probability that Size will be in the range of 10,000 to 12,000 lines of code is:

$$P(10 \leq \text{size} \leq 12) = \int p(x)\, dx$$

where $p(x)$ is the probability-density function in Figure 1a in the main text.

The inverse distribution function, P-1(x), provides values of x that correspond to given values of P(x). Inverse probability-distribution functions are used in Monte Carlo simulation to compute values of x that correspond to randomly selected probability values, P(x).

In practice, you can calculate $P^{-1}(x)$ by table lookup for certain well-defined probability distributions (Z tables for normal distributions, for example) or by sampling techniques such as the Latin Hypercube sampling method.[1]

Monte Carlo simulation is a technique for modeling probabilistic situations that are too complex to solve analytically. Probability distributions are specified for the input variables to the model. A random number generator is used to select independent sample points from the inverse probability distributions for each of the input variables. These sample values are used to compute one point on the specified output distribution(s). Repeating the process a few hundred to a few thousand times produces a histogram that approximates the resultant probability distributions to any desired degree of accuracy.

Until recently, Monte Carlo simulation was the province of modeling specialists. Introduction of PC-based and Macintosh-based simulation packages has made Monte Carlo simulation accessible to anyone who knows statistics and PCs. Two tools for Monte Carlo simulation are @Risk from Palisade Corp. of Los Angeles and Crystal Ball from Decisioneering Corp. of Denver, both of which run in conjunction with a spreadsheet. For the telecom project described in the main text, I used Crystal Ball, to specify probability distributions for the variables in the cells of the spreadsheet, randomly select values from them, and calculate result values according to the spreadsheet equations.

1. [illegible] An Investigation of Uncertainty and Sensitivity Analysis [illegible] 1988, pp. 71-90.


```
Percentage of memory =
100 * [16 * SIZE] / 256          (2)
```

For example, I determined that the percentage of memory used is 93.75 when Size is 15 KLOC. Table 1 shows values of Stor and Time taken from Cocomo.[3] In the first two columns are the values of Stor for various percentages of use. From the table, I interpolated that Stor is approximately 1.55 when the percentage of memory is 93.75.

The last two columns of Table 1 show how execution time affects project effort. Time, which is also dependent on Size, is modeled as

```
Percentage of time = 100 *
[(1/2) * (1/3) * (4 *
SIZE)] / 10                      (3)
```

where 1/2 is the average cycle time in milliseconds for instruction processing on the target processor (five clock ticks at 10 MHz); a third of the object bytes are instructions executed by the main timing loop (an assumption) and the remainder are data cells and exception-handling code; and 4 * Size is the expansion factor from source instructions to object instructions. I then divide Time by 10 ms (the timing constraint) to determine the percentage of time. The percentage of time is 100 when Size is 15 KLOC.

Although, as this analysis shows, the timing constraint dominates the memory constraint, I tracked both factors because the assumption used to derive the percentage of time equation (Time) was not certain and because both Stor and Time affect project effort. In reality, memory could become the dominant factor.

To compute the probable effort for the telecom project, I used Monte Carlo simulation and the Crystal Ball simulation tool from Decisioneering Corp., which randomly selected data points from the inverse probability-distribution functions for Size and Cplx and used the value of Size along with Table 1 to determine values for Time and Stor. The tool then used the values of Size, Cplx, Time, and Stor in the regression equation to compute a point on the probability-density histogram for effort. The tool should repeat this computation at least a few hundred times to produce a reasonable approximation of the probability-density function for estimated effort.

Figure 2 shows the probable effort for the telecom project converted to dollars, because effort was the primary driver of this project's cost. The conversion factor was a loaded salary

of $10,000 per person month, loaded meaning that indirect and overhead costs are included. The right vertical axis indicates the actual number of times the tool computed a given cost. The left vertical axis indicates the probability of that cost occurring, as computed by the ratio of the number of occurrences to total occurrences.

The summation of probabilities up to any given dollar amount is the probability that the project can be completed for that amount of money or less. Table 2 presents some estimated costs and associated probabilities. For example, it is 70 percent probable that the project can be completed for $600,000 or less (60 person months of effort at $10,000 per person month). This cost might involve scheduling six people for 10 months or five people for 12 months.

As illustrated in Figure 2 and Table 2, low complexity and a small product size, with associated small values of Time and Stor, would result in low cost. If the product is large and complex, the resulting cost would be high.

The next issue to face is commitment to a schedule and budget. To distinguish estimates from commitments, I used the equation

$$\text{Commitment} = \text{Estimate} + \text{Contingency}$$

That is, the difference between estimate and commitment is the contingency reserve for the project. In this case, the contingency reserve is for dealing with the impact of uncertainty in source-code size and complexity, and the resulting effects of timing and memory constraints on estimated effort.

In one organization I work with, project teams and management routinely set their development schedules and budgets at 70 percent probability of success, but commit to their customers at 90 percent. The 20 percent difference is a contingency reserve for each project.

Risk mitigation. Boehm recommends avoidance, transfer, and acceptance as potential risk-mitigation strategies.[1] For the telecom project, avoidance techniques might be to buy more memory or a faster processor or to decline the project. Transfer techniques might include implementing the lowest layers of the communications protocol in hardware, placing the top levels of the protocol on a network server, or subcontracting the work to specialists in communication software. Acceptance techniques require that all affected parties (customers, users, managers, developers), publicly acknowledge the risk factors and accept them. They also involve preparing action, contingency, and crisis-management plans for the identified risks.

Action planning. To mitigate the risks of insufficient experience with the target processor, the project manager might provide training for the present staff or hire additional, more experienced personnel as consultants or staff. To deal with the lack of adequate software tools, the manager might acquire more effective tools and provide training. However, he or she would have to evaluate the risk caused by inadequate tools against the risk of insufficient knowledge of the replacement tools.

I used Boehm's Cocomo cost drivers to determine investment strategies for training, consultants, and tools. If training and consultants are expected to lower the effort multiplier for target-machine experience by 10 percent, six percent of this could be invested in training and consultants to

TABLE 2
PROBABLE COST OF EFFORT FOR THE TELECOM PROJECT

Percentile	Cost
50th	$570K
70th	$600K
85th	$667K
95th	$762K

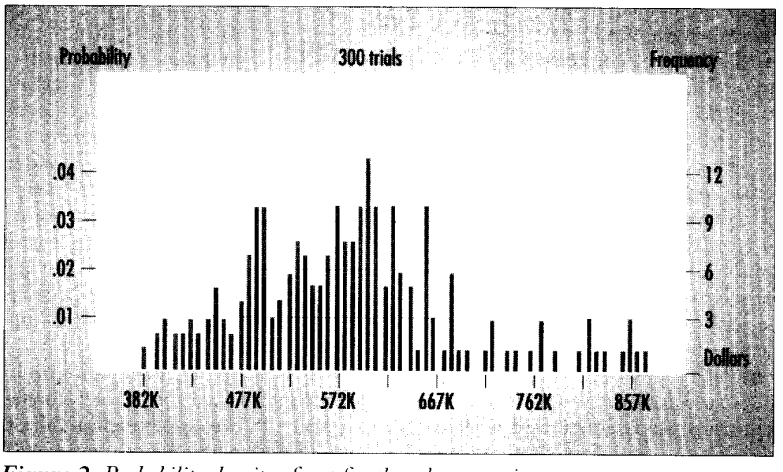


Figure 2. Probability density of cost for the telecom project.

produce a four percent savings in estimated project cost.

Another action plan is to investigate the possibility of buying more memory and/or a faster processor. For the telecom project, the existing processor and memory were provided by the customer and thus required (as in government-furnished equipment), although buying your way out of potential software problems with more and better hardware is sometimes a feasible alternative.

This solution might also involve buying some of the software rather than building it all. However, buying commercial off-the-shelf software is not without risk, especially if you are going to incorporate it into a larger system. The box on the facing page describes some of these risks.

The size and complexity of software in the telecom project were factors for which no immediate actions were apparent: the communication protocols were specified, the team had to use the specified hardware and algorithms, and they could not prioritize requirements and eliminate those that were desirable, but not essential.

Contingency planning. Contingency planning involves preparing a contingency plan, a crisis-management plan, and a crisis-recovery procedure. Contingency plans address the risks not addressed in the action plans. A crisis-management plan is the backup plan to be used if the contingency plan fails to solve a problem within a specified time. A crisis-recovery procedure is invoked when the crisis is over, whether the outcome is positive or negative.

The contingency plan for the telecom project is concerned with controlling the timing budget and memory use on the target processor. It takes into consideration that the probable source size truncates at 15 KLOC, which the code expansion factor of 16 dictates if the major timing loop is to execute in no more than 10 ms.

Thus, preparation of a contingency plan involves

- *Specifying the nature of the potential problem.* For the telecom project this was the effect of memory size and execution time on project effort and schedule.
- *Considering alternative approaches.* For the telecom project, these included building a prototype, using memory overlays, using a faster processor, buying more memory, or pursuing incremental development and monitoring the timing and execution-time budgets. Another approach that is usually considered is to eliminate unessential (desirable but not vital) requirements. However, there are no unessential requirements in a communications protocol.
- *Specifying constraints.* For the telecom project, these were a memory size of 256 Kbytes, an execution time of 10 ms, and the mandatory use of the existing processor and memory.
- *Analyzing alternatives.* Building a prototype would require that the team know how to scale up timing and memory requirements. Using memory overlays would have incurred an unacceptable penalty on execution time. Using a faster processor wasn't possible because use of the current processor was mandatory. Buying more memory wasn't feasible because the processor's address space was limited to 256 Kbytes.
- *Selecting an approach.* Thus, only the last alternative was viable: pursue incremental development and monitor the allocated memory and timing budgets. To do this, the team had to partition the design into a series of builds, allocate memory and timing budgets to each build, and track actual versus budgeted amounts of time and memory for each demonstrated build as the product evolved. A contingency plan was to be invoked when the performance index for actual versus budgeted memory or execution time exceeded a predetermined threshold.

WITH NO SCIENTIFIC BASIS FOR SOFTWARE DESIGN, IT IS HARD TO SCALE UP SIMULATION RESULTS.

In allocating the timing and memory budgets, the team held back a contingency reserve. According to equations 2 and 3, a code size of 15 KLOC would result in 93.75 percent use of memory and 100 percent use of execution time. Backsolving equation 3 showed that developers needed to limit the code size to 13.5 KLOC if they wished to hold 10 percent of the execution time in reserve.

The next step was to form the contingency plan, which involves specifying

- *Risk factors.* In the telecom project, these were the 10-ms timing constraint and the 256-Kbyte memory constraint.
- *Tracking methods.* For the telecom project, these were weekly demonstrations of incremental builds and the monitoring of the memory and execution-time budgets
- *Responsible parties.* For the telecom project, two members of the project team were assigned to monitor performance indices and execute the contingency plan if necessary.
- *Thresholds.* The conditions under which the contingency plan would be invoked. The threshold for the telecom project was a performance index greater than 1.1 for budgeted memory or budgeted execution time.
- *Resource authorizations.* The responsible parties in the telecom project were to be allowed unlimited overtime for two weeks to solve the memory and/or execution-time problem.
- *Constraints.* For the telecom project, the project manager specified that recovery efforts were not to affect the ongoing activities of other project personnel.

Two items in the contingency plan

USING COTS: A DIFFERENT SET OF RISKS

An increasingly popular approach to constructing software systems is to purchase commercial off-the-shelf packages from vendors and integrate them rather than build all the needed components. In many cases this buy-and-integrate strategy is a viable approach to constructing a system. However, many organizations are so enamored with this latest silver bullet that they overlook the inherent risk factors:

- *Integration.* Integrating data formats and communication protocols of various packages can be tricky. In some cases, it may take more effort to integrate the packages than to build the components from scratch. In other cases, subtle "gotchas" in a package may render it useless in your environment, and they may not be apparent until you have already invested substantial effort in the integration.
- *Upgrading.* There are often difficulties in upgrading a vendor's package. The new version may not have the same interface or feature set as the old version. The data formats and communication protocols may be different, and the new version may require more memory and run more slowly than the old version. If you stay with the old version, the vendor will eventually stop supporting it.
- *No source code.* If you need to enhance the system, you may only have the object code. In most cases, vendors are understandably reluctant to provide source code. In the rare instances that they do, the code is usually so difficult to understand that it is very difficult to modify correctly.
- *Vendor failures or buyouts.* What happens to your system if the vendor goes out of business or is bought out? In some cases, purchasers of COTS have made vendors place the source code in escrow, to be available should the vendor's business fail or be acquired by another company. Again, however, having the source code does not guarantee that anyone can understand it well enough to modify it.

I am not advocating that you never buy COTS, of course, merely that you be aware of the risks.

are particularly important: the threshold for initiating the plan (10 percent overrun) and the time limit allotted to fix the problem (two weeks). Because 10 percent of the timing budget is to be withheld, exceeding the performance index for memory or time by less than 10 percent would still yield an acceptable system. A more conservative approach would have been to set the threshold at five percent, while retaining the same 10 percent contingency reserve.

Risk monitoring and contingency planning. To compute the performance indices specified in the contingency plan, the responsible parties compared the actual amount of resources used (time or memory) to the budgeted amount for each incremental build using

$$PI = \Sigma\ (AA/BA)$$

where AA is the actual amount of time or memory required to implement the current build, BA is the cumulative amount of time or memory budgeted for all builds up to and including the current build, and the summation is over all system elements in the current build.

Each weekly build adds functionality to the previous build, so the performance indices track overall growth of time and memory use as the implemented features evolve. Periodically demonstrating implemented capabilities is the only way to accurately track the timing and memory budgets for an evolving software product.

Because software is not a physical entity, there are no physical laws or mathematical theories to guide the development of engineering models that will let us design software to specified levels of reliability, performance, or resource use. The lack of a scientific basis for software design, in terms of traditional engineering parameters, also makes it impossible to scale the results of prototyping and simulation to a full-scale system. This inability, more than any other factor, differentiates software engineering from the traditional engineering disciplines. The only recourse in software is the approach taken in the telecom project: design partitioning, allocation of resource budgets, incremental development, monitoring of budgeted vs. demonstrated values, and contingency plans to control conformance of actual capabilities to requirements.

Crisis management. A crisis is a showstopper. All project effort and resources must be dedicated to resolving the situation. You can define some elements of crisis management, such as the responsible parties and drop-dead date, before the crisis materializes, but you may not be able to formulate the exact details until the crisis occurs.

The elements of crisis management are to

- *Announce and generally publicize the problem.* For the telecom project, a crisis was said to occur if the contingency plan failed to resolve the overrun of memory or timing budget within two weeks. A crisis occurred after the team had implemented half the required functions, overrun the memory budget by 12 percent, and two weeks of contingency actions had not fixed the problem.
- *Assign responsibilities and authorities.* Both of the responsible parties and two other team members stopped all other work to concentrate on the problem. The crisis team had access to all necessary resources, subject to the project manager's approval.
- *Update status frequently.* The project team held daily 15-minute standup meetings at 11:00 am and 6:00 pm.
- *Relax resource constraints.* Management dedicated all needed resources to solving the problem, including flying in two additional target machines. They also provided resources to support personnel work-

TABLE 3
CURRENT STATUS OF THE TELECOM PROJECT

Project activity	Degree of completeness
Requirements designed	27 of 30 designed (90%)
Design elements coded:	75 of 100 coded (75%)
Coded modules tested:	50 of 100 tested (50%)
Tested modules integrated	20 of 100 integrated (20%)
Requirements tested	4 of 30 tested (14%)

TABLE 4
DISTRIBUTION OF TELECOM PROJECT EFFORT

Activity	Percent
Design	17
Coding	26
Code testing	35
Integration	10
Acceptance testing	12

ing around the clock, including catering meals and providing sleeping facilities on site.

♦ *Have project personnel operate in burnout mode.* The crisis team worked as many hours as were humanly possible. All other project personnel were on 24-hour call to assist them until the problem was solved.

♦ *Establish a drop-dead date.* Efforts to resolve the problem were not to continue longer than 30 days. If the problem was not solved by then, marketing and upper management would reevaluate project feasibility. As it turned out, the team resolved the crisis before the 30-day deadline.

♦ *Clear out unessential personnel.* Management requested that all personnel not assigned to the telecom project continue with normal work activities, as long as they did not interfere with the crisis team's work.

One of the most important steps in crisis management is to set a drop-dead date because no one can sustain this kind of effort indefinitely. If the timing problem had not been fixed in 30 days, management would have stopped crisis mode and reconsidered earlier approaches that had been rejected because of project constraints, such as using a different processor or subcontracting the work to telecommunication specialists. They might also have considered moving the upper levels of the protocol to a network server, or even canceling the project altogether.

Crisis recovery. It is important to examine what went wrong, evaluate how the budget and schedule have been affected, and reward key crisis-management personnel.

As part of crisis-recovery, you should

♦ *Conduct a crisis postmortem.* This gives you the opportunity to fix any systemic problems that may have precipitated the crisis and to document any lessons learned. For the telecom project, the postmortem revealed that the design was overly complex in a key area and that a simpler design would have yielded a smaller, faster program. The root cause was the team's overall lack of experience in designing software for the target processor.

♦ *Calculate cost to complete the project.* It is important to know how the crisis has affected the project's budget and schedule. To determine this, I used a technique developed by Karen Pullen of Mitre Corp.,[4] which involves multiplying the expected percentage of total effort for each type of work activity by the actual percentage of completion for each activity. This gave me the current percentage of project completion.

Table 3 shows the status of the telecom project after the crisis. Table 4 summarizes the effort distribution among activities for similar projects. The information in Table 3 indicates an incremental development process; that is, each activity is progressing in parallel with the others. This is consistent with the approach the telecom project team took: Build the product in stages and compare budgeted to actual memory and timing.

Had they taken a waterfall approach, they would have designed all the requirements before beginning coding and completed all coding before beginning acceptance testing. The disadvantage of the waterfall approach is that you don't know if you have an acceptable product until the end of the project. The team would have had to wait too long to find out if the software would fit in available memory and run within an acceptable time — this risk was unacceptable.

Tables 3 and 4 show that the project was 90 percent complete with 17 percent of the estimated project effort (design); 75 percent complete with 26 percent of the effort (coding), and so on. Therefore, the project was 56 percent complete at crisis recovery.

$$90(.17)+75(.26)+50(.35)+20(.10)+14(.12) = 56$$

From project data, I knew that 36 person-months of effort had been expended when the crisis occurred. Therefore, 28 person-months of effort would be required to complete the project, assuming the tasks completed were representative of the remaining tasks. However, the remaining work may be more or less difficult than the work already done, so this assumption must be checked for validity.

Also, I knew that the team had expended six calendar months of a 10-month schedule, with a current staffing level of six people (36/6). Using six people, *and assuming that effort to date was representative of future effort and that no further crises would arise,* the project could be completed in another five months (28/6). This would result in an overall development cycle of 11 months (6+5), plus the time spent on preparing and executing contingency plans and managing the crisis. In the end, the 10-month project was completed in 12 months with 68 person-months of effort. Referring to Figure 2

and Table 2, we see that the project was completed at the 87th percentile of probable effort.

♦ *Update plans, schedules, and work assignments.* Time and resources have been expended on the contingency plan and crisis management, so original project budget and schedule are likely invalid. For the telecom project, mangement added 12 person-months to the budget ($120,000) and extended the project schedule by two months. The contingency plan remained in effect but was not invoked again.

♦ *Compensate workers for extraordinary efforts.* Bonuses and overtime pay are appropriate forms of compensation. However, there is no substitute for resting, regrouping, and recharging. This means time off. The amount of time depends of the level of stress encountered during the crisis. Project managers should factor in that time off when they replan project schedules and assignments. Each member of the telecom project's crisis team was given three days off to recover.

♦ *Formally recognize outstanding performers and their families.* This may include formal letters of commendation, accelerated promotions, and letters to the families of those who worked around the clock. Free dinners and weekend vacations are other ideas. For the telecom project's crisis team, management provided letters of appreciation and dinner certificates.

Many techniques can be used to implement the seven steps of risk management. I have illustrated one approach. Others are certainly possible. Risk management is an ongoing process continually iterated throughout the life of a project; some potential problems never materialize; others materialize and are dealt with; new risks are identified and mitigation strategies are devised as necessary; and some potential problems submerge, only to resurface later. Following the risk-management procedures illustrated here can increase the probability that potential problems will be identified, confronted, and overcome before they become crisis situations. ◆

REFERENCES

1. B. Boehm, *Tutorial: Software Risk Management*; IEEE CS Press, Los Alamitos, Calif., 1989.
2. F. Brooks, *The Mythical Man-Month*, Addison-Wesley, Reading, Mass., 1975.
3. B. Boehm, *Software Engineering Economic*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
4. K. Pullen, "Uncertainty Analysis with Cocomo," *Proc. Cocomo Users Group*, Software Eng. Institute, Pittsburgh, Pa., 1987.

Richard Fairley is the founder and principal associate of Software Engineering Management Associates, Inc. He is also a distinguished visiting professor of software engineering at Drexel University and has more than 20 years experience as university professor, lecturer, and consultant. His research interests are risk management, software systems engineering, project management, cost and schedule estimation, project planning and control, and process improvement.

Fairley received a BS from the University of Missouri and an MS from the University of New Mexico, both in electrical engineering, and a PhD in computer science from the University of California at Los Angeles.

Address questions about this article to Fairley at Software Engineering Management Assoc., PO Box 728, Woodland Park, CO 80866; fax (719) 687-6041.