Part 3

Taking the Measure of the System

8

Identifying and Managing Risk

Software is a risky business. Recall from the previous chapters the spectacular software project failures: the plight of people needing ambulance services in London, aircraft trying to land without radio contact, and NASA probes missing Mars. Software project mangers tend to ignore or minimize risks in their enthusiasm to convince stakeholders to fund or participate in a project. There are great personal rewards for winning project support and little penalty for software project failure. Often a project's first manager is promoted or moved to a new project and is long gone before risks become crises. Neither praise nor blame—and certainly no legal responsibility—devolves back to the original decision maker, but this is commonly accepted business practice. The lack of risk analysis does not lead to a judgment against the vendor whose software project is canceled. If we wish to have the respect and rewards of being professionals, we must accept the responsibility that comes with affecting people's lives.

Software risk management deserves its own chapter because risks transcend the life of projects. The hallmark of successful projects is the ability to identify risks and develop contingency plans to deal with them. Reviews of failed projects typically find that problems would not have become crises if there had been a systematic review of high-risk areas at the start of a project and, more importantly, throughout the life of the project. At the start of each task, circumstances that may prevent the accomplishment of the task, called

Trustworthy Systems Through Quantitative Software Engineering,
by Lawrence Bernstein and C. M. Yuhas

an event, need to be identified. Every event needs a corresponding risk containment and contingency plan.

8.1 RISK POTENTIAL

A risk is the possibility that an undesirable event in the life of a project can happen. Risks involve uncertainty and loss. Events guaranteed to happen are not risks. Events that do not negatively affect the project are not risks. Proactive risk management is the process of trying to minimize the potential bad effects of events.

Risks can affect the project plan and are therefore **project risks**. They can affect the quality of the product, and these are **technical risks**. They can affect the viability of the product, and these are **business risks**. Calculated risk-taking is vital to gain competitive advantage or to pioneer new technology. Rewards await bold and thoughtful risk-takers; obscurity awaits those who take risks blindly and without fallback, also called contingency, plans. Sarbanes-Oxley (SOX) compliance introduces **legal risks** if the elevator injures someone and the cause is traced to the software.

MAGIC NUMBER!

Boehm's hypothesis: Project risks can be resolved or mitigated by addressing them early.

Example: Suppose a project manager is asked to deliver trustworthy software for new high-performance hardware that will be used to control an elevator. This is already problematical because both hardware and software are untried. Table 8.1 provides some risks the project manager identified and classified once the architecture was in place.

With the risks identified, the project manager can keep the project risks and delegate the technical risks to the architect and the business risks to the product manager. All three must determine the likelihood that risk events will occur and estimate potential loss, but at this stage the risks are too general and need to be decomposed into detailed risk events.

TABLE 8.1. Risk Identification in Early Stages

Risks	Project	Product	Business	Legal
Incomplete and fuzzy requirements	X	X	X	X
Schedule too short	X		X	
Not enough staff	X			X
Morale of key staff is poor	X			
Stakeholders are losing interest			X	
Untrustworthy design		X		X
Feature set is not economically viable			X	
Feature set is too large	X			
Technology is immature		X		
Late planned deliveries of hardware and operating system	X		X	
Sarbanes-Oxley (SOX) compliance				X

8.2 RISK MANAGEMENT PARADIGM

"While we can never predict the future with certainty, we can apply structured risk management practices to peek over the horizon at the traps that might be looming, and take actions to minimize the likelihood or impact of these potential problems."[1] The goal of risk management is to make risk-taking a thoughtful and quantitative process. Both short-term and long-term risks need to be considered. The project manager makes sure that the project can succeed if the risk event happens. If the consequences are too great, then a new strategy is needed. One strategy is the early cancellation of a project so that the software organization can fail small and succeed big on some future project, having established their probity.[2]

The Software Engineering Institute (SEI) Risk Management paradigm is shown in Figure 8.1. The paradigm illustrates a set of functions that are identified as continuous activities through the life of the project.

8.3 FUNCTIONS OF RISK MANAGEMENT

Each risk event is examined periodically and formally discussed at project or subproject meetings. This is an important fixed agenda item for these meetings. Risk assessments are assigned as action items and tracked, and new risks are identified and analyzed. The contingency or containment plan for one risk event may yield another risk throughout the project life cycle. The processes for each step of risk management illustrated in Figure 8.1 are enumerated in Table 8.2.

[1] Keil, M., et al. "A Framework for Identifying Software Project Risks," *Communications of the ACM*, 1998, Vol. 41, No. 11, pp. 77–83.

[2] http://www.sei.cmu.edu/programs/sepm/risk/risk.mgmt.overview.html.

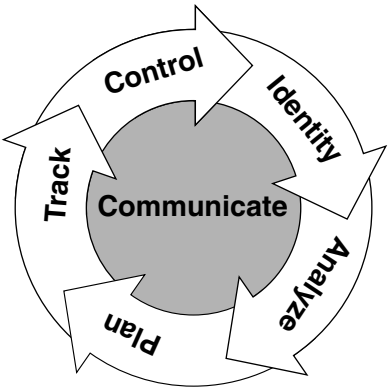

Figure 8.1. *SEI risk management paradigm.*

TABLE 8.2. Processes for Risk Management Steps

Step	Process Description
Identify	Search for and locate risk events before they become crises. Assign risk analysis to a project member as a tracked action item.
Analyze	Transform risk data into decision-making information. Evaluate risk likelihood, loss, exposure impact, and timeframe; classify risks; and prioritize risks.
Plan	Translate risk information into decisions and actions for containment and contingency and implement those actions.
Track	Monitor risk indicators and actions.
Control	Correct for deviations from the risk management plans.
Communicate	Provide information and feedback internal and external to the project on the risk activities, current risks, and emerging risks. Project meetings, progress report, project meeting minutes, and newsletters are effective tools for this communication.

The sophistication of the risk analysis tools depends on the nature of the system, its complexity, the availability of data, cost and schedule constraints, and trustworthy and performance constraints. Modeling is a powerful tool that can predict outcomes and consequences, as well as analyze the interrelationships of subsystems and components. For example, simulation models can analyze failure modes and the consequences of specific risk events. They can be used to conduct "what if" analyses to make engineering tradeoffs.

8.4 RISK ANALYSIS

In 1988, Barry Boehm provided this list of the top-ten software risk items in his seminal Spiral Model paper:

(1) Personnel shortfalls
(2) Unrealistic schedule and budgets
(3) Developing the wrong software functions
(4) Developing the wrong user interface
(5) Gold plating
(6) Continuing stream of requirements changes
(7) Shortfalls in externally furnished components
(8) Shortfalls in externally performed tasks
(9) Real-time performance shortfalls
(10) Straining computer science capabilities

Since then, he re-examined this list of risks in light of newer agile development methods in his 2004 "Guide to the Perplexed" paper. The risks still hold true. Additionally, he and Richard Turner identified five project factors that influence the choice of development methodology: **size**, **complexity**, **dynamism** of the environment and software function, **personnel**, and **culture**. These factors have remained remarkably consistent over time.

We saw that product size dominates the estimates for staffing and schedule. Product size also dominates project risk. Large products require more people, take longer to produce, and are riskier than small ones. The project manager needs to manage these project risk items: project size, project structure, and selection of project technology. Table 8.3 shows how we define size.

Large and huge projects must be decomposed into medium projects with a firm architecture, interface discipline, and formal frequent human communications. Highly structured projects that follow an established development plan are less risky than unstructured projects. The introduction of new technology is always risky.

Based on the risk factors, we can draw Table 8.4 and map the qualitative risk assessment onto a five-ordinal scale (zero through four). This process yields a quantitative estimate of the probability of project success, in which four is likely success and zero is certain failure. Then we map to the range by dividing by five and rounding appropriately.

TABLE 8.3. Project Size Parameters

Size	Developers	Function Points
Small	fewer than 10	150 or fewer
Medium	10 to 25	150 to 750
Large	25 to 200 developers	750 to 6000
Huge	200	Beyond 6000

TABLE 8.4. Risk Analysis

If Size is	And Structure is	And Technology is	Then the Project Risk is	Probability of project success is
Large	High	Low	Low	0.8
Small	High	Low	Lowest	0.9
Large	High	High	Medium	0.4
Small	High	High	Low-med	0.6
Large	Low	Low	High	0.2
Small	Low	Low	Medium	0.4
Large	Low	High	Highest	0.1
Small	Low	High	High	0.2

8.5 CALCULATING RISK

The probability that a **favorable** event (E) in a task will occur is

$$P(E) = m/n$$

where
P is the probability operator and $0 \leq P \leq 1$.
m = is the total number of favorable events.
n = total events.
Risk = $1 - P(E)$ is the probability that an **unfavorable** event will occur.

Risk exposure is the expected value or loss of the risk event, which is calculated by multiplying the risk probability by the cost of the risk event.

$$\text{Risk Exposure (RE)} = (\text{Risk})\,(\text{loss expected if risk happens}).$$

Example: Given that there is a 0.5% probability that a latent fault will execute and lead to failure, and that such failure would cost the customer \$100,000, then

$$\text{RE} = (0.005)(100,000) = \$500.00.$$

To manage this risk, we consider holding a design review. We estimate that the cost for holding this design review in terms of professional time is \$100 and that it will halve number of faults and a new risk exposure (NRE) calculation. Should we hold the review?

$$\text{NRE} = (0.005/2)(100,000) + (0.9975)\,(100) = \$350,$$

where the second term is the cost of the review times the probability that the problem will not happen.

Risk reduction leverage (RRL) is calculated as (RE – NRE)/cost of risk reduction. Therefore, in this example,

$$\text{RRL} = (500 - 350)/100 \text{ or } 1.5.$$

Although a failure may not be avoided, design reviews tend to reduce their occurrence. Even if the RRL = 0, there may be intangible benefits for improving the product, such as making it easier to install, operate, or enhance. One benefit is preventing the erosion of the software structure. Designing for easy maintenance makes room for new modules and enhancements while relieving developers from the harsh resource constraints of fitting into the target computing environment. This effort reduces the risk of implementing future enhancements.

MAGIC NUMBER!

In practice, it is hard to quantify risk probability precisely. Data having a precision greater than its accuracy invites skepticism and mistrust.

A good practice is to qualify the likelihood of a risk event **not** occurring (because it seems easier for designers to think in terms of success rather than failure). A handbook entitled *Software Risk Abatement*[3] used by the U.S. Air Force offers a reasonable way of qualifying both the probability and the impact of risks. First, the probability of the risk event not occurring is categorized as very low, low, medium, high, or very high. Second, the loss to the project for that risk event is estimated using an ordinal scale and Wideband Delphi to do the mapping. Third, the loss is evaluated as negligible, marginal, critical, or catastrophic. Using the second and third subjective measures, an impact/probability matrix is developed to rank order the risk. The matrix should be unique to the problem domain and development organization.

Here are guidelines for assessing risk impact. The definitions of the impact parameters are from Boehm and the quantitative thresholds in Table 8.5 are based on the authors' experience.

Catastrophic: If the risk event occurs, the mission fails or customers refuse to purchase. If the risk event is a project issue, then the cost or schedule overruns exceed 50% of commitments.

[3] http://www.eas.asu.edu/~riskmgmt/intro.html.

TABLE 8.5. Risk Priority Matrix

Impact/Probability	Very High	High	Medium	Low	Very Low
Catastrophic	10	10	6	5	2
Critical	9	9	5	2	0
Marginal	6	5	2	0	0
Negligible	5	2	0	0	0

Critical: If the risk event occurs, the mission might fail or the product might be only a marginal, unprofitable market success. If the risk event is a project issue, then the cost or schedule overruns exceed 30% of commitments.

MAGIC NUMBER!

Reality check: 65% of all software projects suffer **catastrophic** or **critical** risk events.

Marginal: If the risk event occurs, the product may miss a secondary or tertiary objective, or it would yield a return on investment of 10% or less. If the risk event is a project issue, then the cost or schedule overruns exceed 10% of commitments.

Negligible: If the risk event occurs, there would be an inconvenience or non-operational mission impact. Although the early technology adopters would buy the product, the general market would wait for future releases. If the risk event is a project issue, then the cost or schedule overruns exceed 5% of commitments.

Once the analysis is complete, some risks can be chosen and assigned for action. As each is addressed, the next highest priority would be added to the list. The project manager should expect that there would always be action risk studies underway for a medium or large project because, as we have stated several times already, a project with no problems is in deep trouble.

MAGIC NUMBER!

Projects should have ten active risk studies underway, unless they are small projects.

The Constructive Cost Model II (COCOMO II) provides a risk level entry screen for each module and computes risk as the sum of the entries:

$$\text{Total risk} = \text{schedule risk} + \text{product risk} + \text{process risk} + \text{platform risk} + \text{reuse risk}.$$

COCOMO II also provides a heuristic risk assessment capability using the screen in Table 8.6 as input for postarchitecture stage risk assessment.

8.6 USING RISK ASSESSMENT IN PROJECT DEVELOPMENT: THE SPIRAL MODEL

Now that we know some ways to determine the risk and risk exposure for a project and we understand that it is an imprecise science because we are estimating the likelihood of future events, let us examine how we can systematically use these results.

TABLE 8.6. COCOMO II Risk Assessment Capability

Size

	SLOC	% Design Modified	% Code Modified	% Integration Required	Assessment and Assimilation (0%–8%)	Software Understanding (0%–50%)	Unfamiliarity (0–1)
New							
Reused		0	0				
Modified							

Rate each cost driver below from Very Low (VL) to Extra High (EH). For **HELP** on each cost driver, select its name.

	Very Low (VL)	Low (L)	Nominal (N)	High (H)	Very High (VH)	Extra High (EH)
Scale Drivers						
Precedentedness	○ VL	○ L	◉ N	○ H	○ VH	○ XH
Development Flexibility	○ VL	○ L	◉ N	○ H	○ VH	○ XH
Architecture/Risk Resolution	○ VL	○ L	◉ N	○ H	○ VH	○ XH
Team Cohesion	○ VL	○ L	◉ N	○ H	○ VH	○ XH

TABLE 8.6. (Continued)

	Very Low (VL)	Low (L)	Nominal (N)	High (H)	Very High (VH)	Extra High (EH)
Process Maturity	○ VL	○ L	◉ N	○ H	○ VH	○ XH
Product Attributes						
Required Reliability	○ VL	○ L	◉ N	○ H	○ VH	
Database Size		○ L	◉ N	○ H	○ VH	
Product Complexity	○ VL	○ L	◉ N	○ H	○ VH	○ EH
Required Reuse		○ L	◉ N	○ H	○ VH	○ EH
Documentation	○ VL	○ L	◉ N	○ H	○ VH	
Platform Attributes						
Execution Time Constraint			◉ N	○ H	○ VH	○ EH
Main Storage Constraint			◉ N	○ H	○ VH	○ EH
Platform Volatility		○ L	◉ N	○ H	○ VH	
Personnel Attributes						
Analyst Capability	○ VL	○ L	◉ N	○ H	○ VH	
Programmer Capability	○ VL	○ L	◉ N	○ H	○ VH	
Personnel Continuity	○ VL	○ L	◉ N	○ H	○ VH	
Applications Experience	○ VL	○ L	◉ N	○ H	○ VH	
Platform Experience	○ VL	○ L	◉ N	○ H	○ VH	
Language and Toolset Experience	○ VL	○ L	◉ N	○ H	○ VH	
Project Attributes						
Use of Software Tools	○ VL	○ L	◉ N	○ H	○ VH	
Multisite Development	○ VL	○ L	◉ N	○ H	○ VH	○ XH
Required Development Schedul e	○ VL	○ L	◉ N	○ H	○ VH	

Submit Reset

Boehm's Spiral Model proposed a risk-driven software development process. The idea was to identify risk, build prototypes, and evolve software reliability. As originally envisioned, the iterations were typically 6 months to 2 years long. Each stage is a normal development project producing a superset of the prior stage, which will be a subset of the final system. Planning for each successive stage is structured to exploit the experiences of the former stages and to reduce risk factors in the current and future iterations. Although numerous Spiral projects have succeeded splendidly, the Spiral approach has not achieved universal acceptance and has not always produced the results its proponents predict. This model was not the first to discuss incremental design, but it was the first model to explain why iteration matters.

Each development cycle starts with a design goal and list of risk events and ends with a risk assessment and a stakeholders' review of the progress thus far. Analysis and engineering efforts are applied to each phase of the project, with an eye toward the end goal of the project.

For a typical application development, you might build a minimal set of features, without the the user interface. Once there is a usable application, developers add feature sets in increments that correspond to each cycle of the Sprial. Within each cycle there is a mini-Waterfall development and a collection of risk action items. Features are added cycle-by-cycle until the deliverable product is complete.

Typically there will be feature sets left for future releases of the product as the project manager trades off features to meet schedule commitments. A good process is for the project manager to define two sets of features for each release: those that are commitments and those that are goals. The difference between the sum of goals and commitments and commitments alone is the project manager's safety margin. The commitments need to consitutue an economically viable software product.

Using the Spiral Model schedule, and cost estimates are accurate because a risk analysis explicitly addresses budget and schedule problems. The estimates get more realistic as work progresses because the ongoing analysis uses the most current project information. The spiral method copes well with the inevitable changes that face software developers.

Software engineers sometimes become bored and restless with protracted design. The Spiral Model allows coding for the prototype earlier than in a document-driven method. It may be just experimental code to try new algoriothms or to understand performance complexity. This aspect of the Spiral Model deals with the problem of retaining and attracting top-notch people to the project. Using the Spiral Model becomes a risk containment step in itself.

The Sprial Model is ill suited to small and moderate hard-drivng, schedule-focused projects where there is an experienced team in place. When time-to-market is the overriding concern, project managers define lean feature sets and use agile methods.

The Spiral Model is effective in addressing such challenges as rapid development, commercial off-the-shelf (COTS) software integration, new tech-

nologies, and product line management. However, some organizations have experienced difficulties with Spiral development because of overly relaxed controls as compared with document-driven approaches, poorly estimated risks, existing sequential development policies, inflexible financing mechanisms, ingrained cultures, and confusion about what Spiral development is and how to apply it.

Some of the critical success factors for Spiral development are as follows:

(1) Schedules cannot be overly compressed.
(2) Risks must be managed.
(3) Stakeholders must be involved.
(4) The technology must be ready. Combining research with technology adoption and shakedown can be part of a spiral cycle, but adding product development inhibits thoughtful risk assessment. Prototype demonstrations must be in an operationally relevant environment before proceeding to the next cycle.
(5) Requirements must be flexible. Consider an information query and analysis system. The contract was written to require a 1-second maximum response time, which turned out, after 2000 pages of design and documentation were written, to cost $100 million. At that point, a prototype, which would have been created sooner had the requirement been more flexible, showed that a 4-second response time was acceptable and would cost one third as much.
(6) The culture must be supportive. "Buyer, user, and vendor are a team. There is an attitude of partnership, trust and cooperation. There is a presumption of trustworthiness for reputable commercial organizations. Purchase decisions are heavily influenced by personal relationships."[4]
(7) A significant inhibitor is that stakeholders are wary of taking a delivery that only partially satisfies requirements, even though this is their usual software product experience. Retraining at a rapid pace is difficult to tolerate. Good spiral project managers balance training time with the time the release will be used when grouping features into sets.

In one company, field fault densities per 100,000 new or changed thousands source lines of code (NCSLOCS) declined by 62% over 3 years using the Spiral Model, this in an organization already having attained Capability Maturity Model® Level 5 status. This is charted in Figure 8.2, where it is worth noting the industry average of 500 defects per 100K NCSLOC and where the industry best in class lies.

[4] http://www.sei.cmu.edu/cbs/spiral2000/february2000/BoehmSR.html.

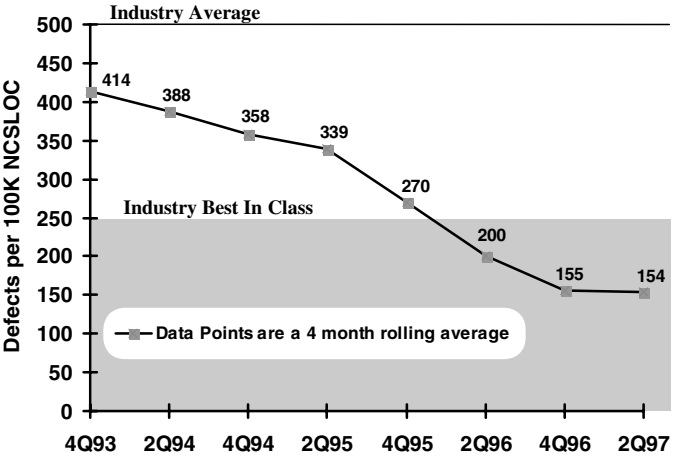

Figure 8.2. *Cumulative field fault density (unpublished data).*

8.7 CONTAINING RISKS

Having the right people and meeting schedules are the most frequently occurring risks to software projects. We will examine the risk containment steps that the project manager can take. Let us return to the risks listed in Table 8.1 in the first section and discuss how to contain each one:

(1) Incomplete and fuzzy requirements
(2) Schedule too short
(3) Not enough staff
(4) Morale of key staff is poor
(5) Stakeholders are losing interest
(6) Untrustworthy design
(7) Feature set is not economically viable
(8) Feature set is too large
(9) Technology is immature
(10) Late planned deliveries of hardware and operating system

8.7.1 Incomplete and Fuzzy Requirements

Different stakeholders in a software development project have their own agenda that often conflicts with the objectives of another stakeholder. For instance, users may require a robust, user-friendly system with many functions that can support their tasks, whereas development team members hope to encounter interesting technical challenges. These differing expectations create fundamental conflicts when simultaneously approached, resulting in unclear or misunderstood requirements.

It may be time consuming and difficult to collect and record all of the required details from all prospective users, resulting in the project team not knowing enough about what is required to complete the project successfully. This may lead to developing a system that cannot be used, mainly because a proper systems analysis to develop a complete and accurate set of requirements has not been performed. Often developers and analysts think of additional capabilities or changes, gold plating, which they think would make the system better and more attractive in their view. These deviations may result in unsatisfied users and unnecessary costs.

Boehm found that a continuous stream of requirement changes is a significant risk. As the users' needs change, so do the requirements of the project. The system will drive changes to business practices that in turn will dictate changes in the requirement. One risk containment approach is to freeze a set of features and a delivery date, but a frozen design does not accommodate changes in business practices. With a frozen design, the developer has little flexibility to change the specifications. Continuous and uncontrolled changes in requirements, however, will inevitably lead to a delay in the project schedule. The software engineer balances these needs.

8.7.2 Schedule Too Short

Yourdon describes the consequences of impossible deadlines and relentless rush, "The key point is to recognize and understand your own motivations at the beginning of a death march project, so that you can make a rational decision to join the team or look elsewhere for your next job. Since many of these projects are initiated during periods of great corporate stress and emotion, rational decisions are not as easy to make as you might think; it's all too easy to be swept away by the emotions of your fellow colleagues or your manager."

Obviously for any important system, other things being equal, the sooner it can be delivered the better. Sometimes circumstances dictate an absolute deadline, like Mother's Day or Y2K. Often some arbitrary deadline, often motivated by political considerations or personal ambition, is proposed, accepted without proper estimating or planning to establish its viability, and then becomes a fixed part of the landscape for managers and developers alike, to be defended at all costs. Where there is a client–contractor relationship, contractors are all too willing to collude with their clients' delusions. Even as the deadline gets nearer and common sense would seem to dictate that it is increasingly unachievable, there is no review and no attempt to change either the deadline or the solution. Shortcuts are taken, and essential processes (such as testing, reviews, problem resolution, and training) are ignored in the rush to complete essential technical tasks. Senior managers rarely take action to prevent this; on the contrary, they are often the prime sources of pressure.

MAGIC NUMBER!

Sixty percent of all projects suffer from compressed schedules.

8.7.3 Not Enough Staff

Product managers, executives, and customers are not impressed with a project manager's strident claims of not enough people to do the job. They want to see analysis supported by credible data. To make matters worse, the various estimating processes can yield different staffing estimates by as much as 50% and there is a highly nonlinear relationship between staffing and schedule time, based on the Rayleigh curve.[5]

Example: Consider a 310 function point project with a second-generation language that expands to 107 instructions per function point.[6] The software lifecycle model tool (SLIM) yields an estimate of 144 staff-months of effort. With eight people assigned to the project, a linear extrapolation will indicate that 18 months are needed, but SLIM shows that only 15 months are required. If the schedule is relaxed to 21 months, then four people are needed. If the fully loaded cost of a developer, including salary, benefits, and general and administrative costs is \$70/hour or \$12,000 per month, the conventional schedule will cost with linear extrapolation: (18 months)(8 developers/month)(\$12,000) = \$172,800. Similar calculations show results for the nominal schedule and the relaxed schedule in Table 8.7.

TABLE 8.7. Staffing vs. Schedule Time

	Developers	Schedule (Months)	Cost (\$12k/SM)
Conventional Linear Extrapolation	8	18	172,800
Nominal Schedule	8	15	144,000
Relaxed Schedule	4	21	100,800

[5] Putnam, Lawrence H. and Myers, Ware. *Industrial Strength Software: Effective Management Using Measurement*, IEEE Computer Society Press, Piscataway, NJ, 1997, Ch. 18 for discussion of SLIM and Rayleigh curve.

[6] http://www.spr.com/products/programming.shtm.

Considering that most project cost estimates are low by a factor of 2 or 3, these costs estimates, and remember that is all they are, are reasonably close. The SLIM tool exaggerates the effects of compressing schedules, but the point is valid and within reasonable estimating bounds. Without this analysis, the cost estimates become a matter of opinion. Bosses tend to trust their own opinions unless they are given the facts.

In an atmosphere of internal or external pressure to cut or control costs, cost reduction becomes the single-minded goal of the boss. Low development cost is deemed essential to gaining project approval or the prime criterion in bid selection. Just as there is resistance to changing deadlines, there is often a natural reluctance to cancel projects with runaway costs. Investment already made in the project feeds the escalation cycle and results in throwing more money after bad, rather than a re-evaluation of the return on continuing investment.

8.7.4 Morale of Key Staff Is Poor

Experienced project managers understand that containing risks is an effective way of keeping the morale of the staff high. Project managers who do not actively manage risks before they become crises rely on heroic efforts to keep the project on track, which burns out the staff and lowers morale. Developers will tend to stay to complete the current feature set and then leave the project when it is released.

Studies of risk management show that as the number of projects managed increases, the risk factors "unrealistic schedules and budgets" and "misunderstanding the requirements" occur less often. Managers can learn. No similar conclusions are possible for the other risk factors. Middle managers need to nurture the development of project mangers with formal training and education in software project management skills. Assigning novice project managers to small subprojects is a good first step. These might be leadership in developing a feature set requiring the coordinated work of two to four people. This builds confidence and esteem from colleagues. A proven record of accomplishment of project success is the best recommendation for a project manager. Having active project managers participate in reviews of other projects is effective in giving the project manager insight without having to suffer "trial by error" learning. It also encourages technology transfer between projects.

Software developers need to feel that they are growing in their field. They fear becoming obsolete and like to try new approaches and new technology. By providing life-long education and discretionary money for exploration, executives will manage this morale risk. These personal goals and values contrast with those of the executive. Once people are comfortable that they are being paid fairly and are satisfied with their standard of living, then different reward systems seem to motivate people in different jobs:

(1) CEOs seek compensation.
(2) Managers seek promotions.
(3) Salesmen seek commissions.
(4) Administrators seek appreciation.
(5) Software developers seek opportunities to tinker.

8.7.4.1 Agile Methods Agile methods are a risk containment process for the restless programmer. Agile methods encourage collaboration with people they respect and with whom they can grow.

Microsoft Program Manager David Anderson started as a game developer in the 1980s; he was involved in emerging agile techniques such as pair programming and short lead times. He later went on to help develop feature-driven development and comments:

> Agile is really catching on all over the company, within product groups and the internal IT organization. We're seeing . . . experiments with test-driven development and program managers running Scrum-like 30-day sprints and daily stand-up meetings. There's a growing community of agile believers . . . program managers at Microsoft don't have direct reports, so they don't manage programmers – development managers do that. However, both program managers and development managers need to be technical. It's all about respect. Developers need to respect the technical ability of their leader. Without respect, you cannot lead; and without leadership, software projects tend to fail . . . I'm not a believer in measuring individuals by their code production. Developing software is a team sport; it requires interaction and mutual support across the team. It's knowledge work and is best done in an environment of knowledge sharing. When you reward people for individual effort relative to their peers, you encourage them to hoard knowledge rather than share it. The manager should be measured by the productivity of the team, not the individual team members for their individual efforts . . . I reward people to learn and share. It's behavior compatible with team success . . . People won't follow unless they see and understand where they're going, and see confidence and resilience in their leadership . . . strong management is essential. Management must be prepared to instill and enforce discipline. Discipline in software development is what delivers high-quality, low-defect code.[7]

Philippe Kruchten, the inventor of the "4 + 1" architecture model, points out that the agile process "sweet spot" is for small teams of 10 to 15 developers who are colocated and communicate verbally as opposed to with documents. The customer representative on site is domain literate and empowered to make decisions. The projects have short life cycles of weeks or months (not years), and they use powerful development tools including automated rapid development, frequent (usually daily) builds, and automated test drivers and regression test suites.

[7] http://click.sd.email-publisher.com/maac2NEabc3WmbdnjEbb/.

MAGIC NUMBER!

Agile processes are tuned for ten-person projects.

8.7.4.2 Best for Small and Medium Applications Agile methods seem to work best for small- and medium-sized business applications rather than for high-performance embedded and real-time systems. New developments rather than maintenance projects are well suited to the agile methods. But maintenance, high performance, and large 200-staff projects are not going away. These large teams are geographically distributed, have few development tools, and an empowered customer is never present; they rely on documents for communication and strive to minimize changes. With a component-based architecture, a large project can be broken into a collection of medium projects and then decamped again into small projects. The architecture relies on a strong interface language that restricts how components exchange data and control; this constraint may be relaxed by skilled gurus, but it is inadvisable to do so. Then agile methods can be happily used to build the small components, and then their source code is transferred to the manufacturing team.

The manufacturers build a system following the instructions of the integration team. The components are recompiled and then linked, loaded, and regression tested. The integration team then performs functional tests based on use cases, reliability, and stress tests. If even more trustworthiness is needed, the system can be handed to a specialized team of diabolic testers. The integration team verifies that the system is ready for release. Even after release they may continue to test the release certifying the ability of the release to handle anticipated customer demands as the release is deployed. In this scheme, developers are related to agile method-based teams to build components. Then the realization for the software system becomes an integration process. Various software foundries build software components using agile methods. They feed their source code to carefully controlled libraries, and the technical integration team gives build instructions to the software manufactures who compile the source, load it into a system test machine, link the components, and run regression tests. Then the integration team tests the system to make sure it is ready for release. This **Compile, Link, and Test** metaphor for building software systems requires substantial support and emphasis on noncoding activities.

MAGIC NUMBER!

Five percent of a large project's staff is needed for communication among small agile teams, preparing system-wide documentation and packaging software for release.

8.7.5 Stakeholders Are Losing Interest

Users become more involved when there is a steering committee in place and they are informed of the impact of scope changes. They need to be welcome at all project meetings and have access to project information. The project manager who builds trust with the stakeholders increases the productivity of the organization and is more apt to build products that will be bought and used. The steering committee's goal is to stabilize requirements and specifications and to select feature sets proposed by the project manager.

When all stakeholders from customer to user to developer to supplier identify themselves as a member of the project before they identify their organization or role, the project manager will know that the stakeholders are not losing interest. It is a warning sign of trouble when stakeholders consistently miss steering meetings or send ineffective delegates. The risk containment action is to visit the stakeholder one-on-one and face-to-face to discuss their concerns. If this is impossible, then the project manager needs to find new stakeholders. Experienced project managers sometimes "fire a customer." Project managers need to report to their bosses and project team members the results of their periodic and frequent meetings with their customers. Another risk containment strategy is for the project manager to visit the users' workstations and see how the system is operating. On-site observations build confidence in the customer's staff that you care about them.

8.7.6 Untrustworthy Design

Requirements creep can subvert assumptions that underpin the architecture. New requirements need to be carefully controlled. Loose talk by developers quickly leads to commitments in the minds of customers. Although it is important to encourage open and complete communication between developers and users, it is just as important to be clear that only the project manager can make project commitments. Do everything to ensure common understanding of requirements among stakeholders.

Reuse can be a key issue. Systematic and well-managed reuse of elements of previous systems/solutions can bring great benefits, but without proper understanding and analysis, they can be high risk. Wrong decisions on technology infrastructure can have massively damaging effects. Distributed computing rather than large computers requires middleware and complex programming to share workloads, which adds risk to overcommitted software activities.

8.7.7 Feature Set Is Not Economically Viable

The risk of developing the wrong software functions and of doing the features wrong is reduced when the project is divided into separately controlled subprojects and requirements specifications are stabilized.

8.7.8 Feature Set Is Too Large

This is known as the Big Bang implementation. The attempt to leapfrog to a system that delivers the expected functionality and benefits in a single-shot implementation is fraught with risk. With complex systems, phased implementation of each phase delivers some useful functionality and builds on what already exists. This offers a controlled approach to dealing with risk. The Spiral Model offers one way to contain this risk. The overall effort can be assessed based on results to date, and it becomes easier to spot trouble earlier and to adjust overall effort if necessary. This approach also gives users a flavor of what the system can do for them, can generate enthusiasm and support, and can reduce resistance. On system cutover from a legacy process or system, a tried and true fallback system provides risk containment. Of course, the necessary cutover procedures with staff trained in their use are always important, but it is even more so when the Big Bang is attempted.

8.7.9 Technology Is Immature

Blind optimism that novel technology solutions are achievable is contagious and dangerous. Developers by their nature tend to innate technological optimism, and this is not always counterbalanced by management realism. Management is often technically obsolete. High levels of system complexity, ambition, and innovation are not recognized, and necessary steps to manage them are not taken. Many projects fail because they seem to be always adopting the latest and greatest before they capitalize on their last investment. Tool suppliers know to sell their new products to the developers and let them champion the tool to their management.

The prospect of enormous benefits and the splendor of technical achievement mask the scale of the challenge and the risks of failure. Experience with smaller simpler systems, or in other domains, makes software people believe that they are qualified to take a leap of faith. The required knowledge for attempting such a breakthrough in technology does not exist, and the need to

extend budgets and schedules to allow for research and experiment to acquire that knowledge is critical.

How can project managers choose technologies? Refusing all technology changes leads to obsolescence. Need, expertise, and experience influence a technology choice. To manage the risk, the project manager assigns specific resources to explore the technology, then tries it in a few isolated cases in the environment of the project, and finally trains the full staff to adopt the chosen technology.

Case Study: The Case of Trouble in Paradise

At Paradise Software, C++ is de rigueur, but lately there has been trouble in Paradise. Many field problems have been traced to pointer arithmetic errors. The new kid proposes shifting to JAVA. Some developers like the idea because they can learn a new language and programming environment. It will make them more attractive should they ever need to find a new job, not that they are eager to leave Paradise.

The project manager is old and battle-scarred. He lost 6 months of production from every staff member when they converted from C to C++ 10 years ago. After 18 months the loss was recovered, but it was a difficult 18 months. True, his software shop did gain a reputation for world-class, low-defect, on-time software afterward. JAVA will probably yield higher productivity eventually, but not immediately.

The project manager reflects with pain on the earlier transition and sighs. He knows JAVA produces better code, but he wonders if it yields acceptable application performance and if the tool base is stable. Is this the right time to switch from an old technology to a new one? He decides to hedge his bets this time around.

The project manager agrees to set up a pilot program of no more than three people for no longer than 6 months. Their job is to qualify, quantify, and formally report the performance impact of the new language and the quality of the support tools. In addition, the project manager does his homework researching the experience of other projects with JAVA and finds this amazing graph, Figure 8.3.

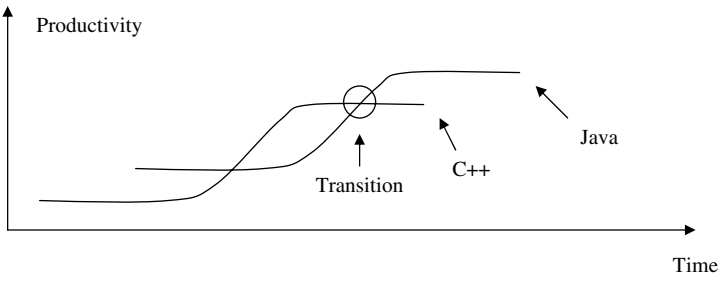

Figure 8.3. *Productivity vs. learning curve re C++, JAVA.*

Conclusion: The pilot program reports are satisfactory, and the project manger sets up and pays for trials with training and technology experts assigned to work in the trials. Once the trials are complete, a decision for project-wide deployment may be reached. If at any point the risk of adopting the new technology is too great, the technology can be dropped. The new kid thinks Paradise is a pretty good place to work. They listen.

MAGIC NUMBER!

One pilot plus three trials equals informed decisions.

8.7.10 Late Planned Deliveries of Hardware and Operating System

Alerts to potential problems may come from a variety of sources: past experience, warnings from knowledgeable sources, feedback from reviews, jeopardy reports, observation of events, or just plain common sense. Deafness to such alerts may afflict any stakeholder group including executives, customers, product management, project management, and developers. When those capable of sounding the alert observe that deafness or punishment is the usual outcome, then the alert will be sounded less frequently.

Dysfunctional relationships among stakeholder groups can develop. These key stakeholder groups may include executives, middle managers, developers, the project sponsors, or the system users. Agile methods and the WinWin Spiral Model demand that all stakeholder groups be integrated into the development process. Everyone needs an effective voice to express both their needs and their knowledge. Poor intergroup communication increases risk, and success relies on establishing stakeholder ownership and commitment. Examples of dysfunctional relationships between groups include:

(1) "Range Chicken," a concept that originated during missile tests and carries over to enterprises that require the cooperation of several independent organizations. As launch time or due dates draw near, organizations having problems delay reporting their problem in the hope that they will find a fix before they must confess to not being ready. They do not want to be blamed for delays.
(2) Unwillingness of executives to understand issues, get involved and committed, organize for the systems effort, and undertake properly informed decision making.
(3) Resistance of middle management to relate concurrent problems or changes within the enterprise and to understand and deal with their

interactions. For example, computer systems cannot automate chaotic processes.

(4) Unreasonable pressure by customers or senior management on development schedules.

(5) Lack of attention to staff concerns of both developers and users exacerbated by overspending of scarce resources on glossy brochures, management consultants, and corporate image, or by perceptions that it is lying to the public. Several systems have failed when union workers perceived that the objective of the system was to eliminate their jobs. They made sure it did not work. People want software systems that enhance their positions, not replace them.

(6) A climate of fear, blame, and low morale among developers, vendors, or users.

(7) A culture of strong internal politics, improper relationships, and vested interests among customer agents, developers, or suppliers.

8.8 MANAGE THE COST RISK TO AVOID OUTSOURCING

Although project teams are effective in determining customer needs and satisfying them, they are not efficient in their use of resources. There is much duplication between teams in the areas of tool selection, technology evaluation, process design, software manufacturing, computer administration for both clients and servers, and in setting up and operating system test environments.

MAGIC NUMBER!

Up to 20% of the budget for large multiproject organizations is wasted in redundant activities or lost because they do not receive volume discounts for hardware or software COTS tools and components.[8]

Software shops can gain the efficiency they need to be competitive and avoid the outsourcing risk by forming some functional departments to serve across all projects. Introducing centralized functional departments often fails, so it must be done with great care and middle management buy-in. Choosing the tasks to centralize and full commitment by the entire leadership team are

[8] Internal Bell Laboratories study of a 2000-person, 150-project software shop led by Bernstein.

vital. On the other hand, centralized departments too often become bureaucratic and lose the sense that their mission is to serve the projects. To manage this risk, rotate middle and first-line managers through the functional organization periodically.

MAGIC NUMBER!

Rotate managers in functional organizations to project organizations every 2 to 3 years. Staying longer tends to make the functional organization less responsive to the current needs of the projects. Fresh managers update the centralized activities.

8.8.1 Technology Selection

Centralized functional organizations are more effective at choosing technology because uncontrolled diversity often leads to systems that cannot work together. Components cannot be shared among teams, and basic functions such as systems and network management are duplicated. Even worse, any resemblance among systems administration for different product lines is purely coincidental. Without being able to share the platform rules, tools, and assets, it is näive to think that application areas can be shared. Each project loses skilled people to meeting with suppliers, selecting technology, and training new team members on their selected technology.

8.8.2 Tools

When every project team does this, it duplicates the efforts of the others. It is difficult to obtain leveraged purchases on favorable terms from tool suppliers when each team buys for themselves.

8.8.3 Software Manufacturing

A duplication of effort to control changes and build releases across projects without adherence to cross-project standards leads to difficult system integration. Often products must work on different physical computers from the same manufacturer and with the same configuration because the software executables are arbitrarily different. Each team has its own approach to system builds, and there are no opportunities for economies of scale This increases testing costs as well as cost to the customer. Software manufacturing activities include trouble tracking, system builds, configuration control, and release packaging.

Centralize these functions, and include a trained set of program administrators whose job is to adhere to standard processes so that the programmers learn to trust the professional software builders.

8.8.4 Integration, Reliability, and Stress Testing

This area should come under centralized functional management, and the system testers should be placed on loan to product teams for specific tasks. System test processes and tools should be standardized. Subject matter experts need to remain with the project teams, but software testing technologists and their tools may be centralized. The new concept of software component foundry and "Compile, Link, and Test" software factory is an effective way to gain economies of scale, in which component is a technical definition constraining the characteristics of a software module.

8.8.5 Computer Facilities

The administration of the client and server computers spread throughout the organization is a hidden expense. Volume purchases are difficult to make, and systems administrators with the knack of keeping the computers operating are not shared. There is significant wasted space and duplicate machines. Inadequate desktop tools can impact the entire development community and were one root cause of delays in several projects.

8.8.6 Human Interaction Design and Documentation

Almost 25% of the total staff is devoted to ease of use and user documentation. Standard approaches to doing the design, producing, and controlling the documents can be centralized. Skilled human performance engineers and technical writers can be loaned to projects as needed, which improves their morale because they are not lost in a program design organization and have a path for career growth.

All six of these functions can be separated from the project teams and managed in a single integrated support department. This matrix organization works when funding for their funding is not within each project team. The centralized team has its own budget with usage-based charge-backs to the project teams. Cost under-recoveries reduce profits for all projects inversely proportional to their use of the centralized activities, and over-recoveries increase project profits based on their use of centralized activities. They are a cost center; the project teams are the profit center.

This centralization would not only improve productivity but also would reduce time to market and improve quality. Many project teams cannot make incremental builds. They always deliver their whole product package to the customer. This is costly to both the projects and the customers. It is a major customer dissatisfaction as they expect products from the same software shop to "look and feel" the same, share file storage and networks, and have a

common system administration. The centralized team may in turn sell their services and tools outside the organization, but only through a project established for that purpose.

This approach is the foundation for establishing a climate for reuse that leads to a significant increase in productivity. The essence of the these standards is to define and implement a set of engineering rules that make it possible to practically integrate products into customer-specified solutions through reuse and commonality of processes, development methodology, and tools. The engineering rules encompass four categories: software architecture and platform; environment and tools; integrated product offers, and documentation, including sales-ware.

Case Study: The Case of the Perpetual Pendulum

A classic management problem is how to determine the best structure for an organization that must be nimble, yet at the same time stable enough to get the work done. Change renews and invigorates, so long as it does not happen too often. This risk management technique prevents the ossification of a company. Remember *The Case of the Well-Shod Management* in Chapter 6; this is the next episode.

The manager found that 20% more savings were possible by adhering to engineering rules. This amounts to a saving of 40 people $\times$ 0.20 = 8 people or another $1.3 million annually. This leads to a 50% cost reduction.

Arriving back at corporate headquarters, the manager delivers this good news. The Executive Council has been infected with "outsourcing fever" ever since their CIO attended a 2-day seminar. Not only do they applaud the news, but they leap to send every last employee out the door by outsourcing everything.

The manager pauses to reflect that successful outsourcing by a company requires dedicated company project managers and software engineers, special testing, travel, communications, and equipment costs.

Conclusion: This letter to the Executive Council by the project manager successfully argued against outsourcing this project. The project is still staffed domestically today.

To: Vice President, Operations Support Systems Business Unit

There is no need to outsource to gain competitive advantage. Our development team is in place, is skilled, and has gained extensive problem domain knowledge. There is significant opportunity to further improve our productivity and quality while reducing time-to-market. Introducing centralized functional organizations is a vital part of this effort. Action is being taken to obtain an immediate 10% elimination of redundant effort, which is expected to grow to 20% in 3 years. Specific areas that will to be addressed include:

(1) Engineering rules conformity
(2) Reuse
(3) Platform and library use across projects
(4) Use of a software fault-tolerant library to reduce testing efforts and produce a better production product
(5) Software project management training
(6) Move to "buy instead of build" culture and create a "link, compile and test" software shop
(7) Migrate to JAVA and JAVA platforms
(8) Set a 40% design simplification goal
(9) Reward developers for writing less code per feature

Yours truly,
The Project Manager

8.9 SOFTWARE PROJECT MANAGEMENT AUDITS

Management malpractice is a serious risk to project success. When a software project is in trouble, a software project audit can help. Audits are for the project manager and not for higher management. The audit team needs to gain the trust of the software developers. Audits should be a regular action taken early in the development process before implementation begins. McDonald writes that with this approach, "The audit became relatively painless for the project team; it is likely to cause a much more positive change in the software development environment. . . ."

Software project management audits address the risk of management malpractice and pressure the project leaders to create a realistic project plan early in the development cycle. Audits crisply identify the highest project risks and estimate their probability of failure.

The audit is requested by the project manager. McDonald recommends, and we agree, that the best ". . . time for conducting a project management audit is during the definition phase of a project, shortly after the business case or contract has been baselined, a solid project plan has been developed, the architecture has been baselined and at least some of the detailed requirements have been developed." But an audit can be useful anytime. A software project management audit includes a review of the business case, the development plan, the architecture, and the requirements specification. If a prototype is available, the audit team examines it also. They visit the development team in their workplace and at their desks. They observe activity in the test laboratories. Fundamental to the audit is one-on-one interviews with a cross section of the project team by an independent set of experienced managers who have managed similar projects.

The audit produces feedback only to the project team; they may in turn comment on it and forward it to their executives or customers as they see fit. It is their intellectual property.

Audits have contributed to project success when the project manager embraces them and uses the results constructively. The project manager and the development team members get the most benefit when they view the auditors as fellow developers interested in their success, not as part of an inquisition.

8.10 RUNNING AN AUDIT

Before an audit takes place, the project leaders spend time learning about the project and creating their measurable operational value and development plan. The project manager and the audit team leader jointly clarify the scope and the objectives of the audit and schedule it. It is not a surprise visit by an outside set of professional auditors. The project manager helps the audit team leader recruit audit team members and communicates audit findings with follow-up actions to the entire project team.

The audit team leader is an experienced software project manager who has previously participated in audits and will choose three or four other audit team members, who are skilled project managers. The audit team leader structures a detailed time schedule for the audit and a questionnaire. The project manager approves these.

McDonald reports the results of 21 audits of medium- to large-scale software projects. He was on each audit team. As predicted by Boehm, project schedule and requirements management issues were the most frequently cited risks.

8.11 RISKS WITH RISK MANAGEMENT

All software engineering processes have advantages and disadvantages. Risk management is no exception. The project manager reaches a balance between control and productivity so that the development staff can be effective and efficient. Risk management is a worthwhile process, but it can intimidate an organization when risks are "overestimated." Then too many resources may be diverted to contingency planning and containment steps and too much money may be spent on the most vocal risk identifier's pet part of the project. Experienced project managers are needed to contain this risk. When software shops approach schedule deadlines, a nervous tension close to panic is observed. Managers seek delays identifying all possible risks. The project managers use the top-ten problem list to manage these risks and judge the state of the project by examining the nature of the problems. The mantra at this late stage switches from "What's the problem?" to "Plan on Success." This requires delicate and experienced judgment at a critical phase. On one project, the reports of problems reached such a crescendo that the customer suggested that the project be delayed. The project manager cautioned not to revise the project

plan until after the next critical milestone. Then, if necessary, have a critical schedule review. In addition, minor adjustments were made in staff assignments setting up a task force of testers and developers to help in component testing. Happily the dates were made, and there was no need for the review. Keeping firm in times of project turmoil is essential, but not easy.

Risks may be dismissed and their impact underestimated, which gives the stakeholders a false sense of security. This risk can be contained by the use of project audits. Software shops that underestimate risks become victims of reactive crisis management. Blame must be avoided when a project gets into this state. Public recognition and appreciation for those identifying problems is necessary to establish the open and trusting project environment needed to deal with crises.

Choosing projects that minimize all risk can lead to dull projects and drive away the gurus. These projects are typically less profitable. Risk management can be expensive and, as with all processes, needs to be managed to make sure it is finding real problems, inspiring quality solutions, closing risk items, and not breaking the bank.

8.12 PROBLEMS

8.12.1 Given that there is a 0.1% probability that a latent software fault will execute and lead to failure, and if a failure occurs the customer would lose $100,000, what is the RE?

8.12.2 To manage the risk found in Problem 8.12.1, we consider a holding a formal design review. How much can the project manager afford to spend on the design review and break even? (Hint: Break even means the RE without design reviews equals the RE with design reviews.)

8.12.3 List four principle requirements risks as defined by Boehm in the development of software products.

8.12.4 You are the project manager for a customer resource management system. Your sales force tells you that the architecture impacts their ability to sell your software product. Here is the prospectus:

Department stores, software companies, and utility companies all have call centers that handle customer complaints. Your primary customer employs 1000 agents. Call center managers have the authority to hire and fire agents and can purchase incidental equipment for the operation of the center. Purchases of more than $100,000 require corporate approval including a review by the CIO. The CIO is charged with reducing information technology costs and the number of suppliers. Typically an agent uses a predefined script to capture the customer's problem. Once the problem is defined, it is resolved or handed off to a second-tier expert. The agents must strictly follow the script

that can resolve 50% of the problems. For example, if a customer claims that they have already paid a bill, the agent asks for the invoice number and checks the accounting database. If the customer's payment has been recorded in the time since the bill was mailed, the agent cancels the bill; if this is a first customer complaint and the bill is less than $10, the agent forgives the charge. The system must be generally available in 9 months.

An existing customer for your software company badly needs such a system to automate these business tasks. They want to purchase the system in 6 months. This customer has had serious problems with another system they recently bought from your company, and their CEO has formally complained to your CEO about late delivery, "obvious" bugs, and poor support.

For the CRM system, your staff of five developers prototype a new speaker-independent voice recognition system, using JAVA, and find that customers prefer the clarity and patience of the computer to that of many agents. Reliable and consistently friendly agents that exercise good judgment are hard to find and train at the wages companies are willing to pay. An agent is paid $30,000 yearly, and the overhead is twice the salary.

Based on the prototype, you size the project at 50 function points. Desktop computers equipped with voice recognition equipment, communication hardware, and platform software costs $10,000. These computers can replace one agent.

Server computers can share the new speaker-independent voice recognition equipment, and communications hardware and platform software costs $120,000. Each server can replace 300 agents, which results in a cost of $4000 per agent.

Financial times are hard, and your likely first customer, the CIO, is insisting that new systems yield a 2-month or less break-even time where the cost of money is not considered because interest rates are 1% annually. The cost of developing the software is $800,000. This is true for either the server or the desktop solution.

Identify the top-ten risk events that you need to put on the agenda for the next CRM project meeting:

8.12.5 You are asked to build a system that is priced to yield a profit of 10% of the cost of the system in 3 years if it is successfully deployed. Use a constant interest rate of 5%. (Hint: To discount the cash flow, compute today's value of future money by using this formula:

$$\mathbf{NPV} = \mathrm{CF} / (1 + \mathrm{IR})^{n},$$

where
NPV = Net present value
CF = Cash flow
IR = Interest rate (3%, for example)
n = Number of years

a. What is your return on investment in the third year?
b. The schedule is known to be tight, and the nominal schedule is estimated to be 44 months. A schedule with 95% confidence is estimated to be 60 months. Estimate the probability that the system will be delivered on the 3-year schedule if all delays are assumed to be uniformly distributed.
c. What is the expected profitability of the system?

8.12.6 Recall the case study about Ajax Transporters, Inc., which manufactures exactly one size of one model of one product, the Ajax Personal Transporter, in Chapter 7.

You work for Amber Consulting, Inc., and your group at Amber has just been given the job of making the change to Ajax's system, but without changing the COTS product currently in use. You are required to keep the COTS subsystems current.

Define five risks for this project, and calculate a risk exposure for each. State your estimate of risk likelihood and risk cost.

8.12.7 You are the project manager for a large transaction system with access to a large database. You estimate that the development will take 3 years, and you propose to provide annual releases. You estimate that you will need 40 to 60 people per year, and you request a budget for 55 people. Your boss challenges the accuracy of your estimates. You respond that they are accurate to about +20%. He reduces your budget request by 11 people. What is your response?

a. Accept the change and try harder.
b. Look for a new job.
c. Point out that just as you may have estimated high, you may have estimated low and that you might be overbudget by 11 people.

BIBLIOBRAPHY

Addison, Tom and Vallabh, Seema. "Controlling Software Project Risks—an Empirical Study of Methods used by Experienced Project Managers," *Proceedings of SAICSIT*, 2002.

Department of the Air Force, Software Technology Support Center. "Guidelines for Successful Acquisition and Management of Software Intensive Systems: Weapon Systems, Command and Control Systems," *Management Information Systems*, Vol. 1, Version 1.1, Department of the Air Force, Software Technology Support Center, Salt Lake City, UT, 1995.

Asaravala, Amit. "Managing at Microsoft," *SD PEOPLE & PROJECTS*, Jan. 2005.

Bernstein, L. "Software Project Management Audits," *Journal of Systems and Software*, Vol. 2, 1981, pp. 281–287.

Boehm, Barry. *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981, Chs. 19 and 20.

Boehm, Barry. "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, May 1988, pp. 61–72.

Boehm, Barry. *Software Risk Management*, IEEE Computer Society Press, New York, 1989.

Boehm, B. and Ross, R. "Theory-W Software Project Management Principles and Examples," *IEEE Transactions on Software Engineering*, Vol. 15, 1989, pp. 902–916.

Boehm, Barry W. "Software Risk Management: Principles and Practices." *IEEE Software*, Vol. 8, Jan. 1991, pp. 32–41.

Boehm, Barry, et al. *Software Cost Estimation with COCOMO II*, Prentice-Hall, Englewood Cliffs, NJ, 2000, p. 403.

Buckle, J. K. *Managing Software Projects*, American Elsevier, New York, 1977.

Endres, Albert and Rombach, Dieter. *A Handbook of Software and Systems Engineering—Empirical Observations*, *Laws and Theorems*, Pearson Addison Wesley, Reading, MA, 2003, p. 201.

Keil, M., et al. "A Framework for Identifying Software Project Risks, *Communications of the ACM*, Vol. 41, 1998, pp. 77–83.

McDonald, James. "Software project management audits—update and experience report," *The Journal of Systems and Software*, Vol. 64, 2002, pp. 247–255.

Putnam, Lawrence H. and Myers, Ware. *Industrial Strength Software: Effective Management Using Measurement*, IEEE Computer Society Press, New York, 1997, Ch. 18 for discussion of SLIM and Rayleigh curve.

Yourdon, Edward Nash. *Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects*, Prentice-Hall, Englewood, Cliffs, NJ, 1997.