End [illegible]

SafeHome

Outsourcing

The scene: Meeting room at CPI Corporation.

The players: Mal Golden, senior manager, product development; Lee Warren, engineering manager; Joe Camalleri, executive VP, business development; Doug Miller, project manager, software engineering.

The conversation:

Mal: We're considering outsourcing the *SafeHome* software engineering portion of the product.

Doug (shocked): When did this happen?

Lee: We got a quote from an offshore developer. It comes in at 30 percent below what your group seems to believe it will cost. Here. [Hands the quote to Doug who reads it.]

Mal: As you know, Doug, we're trying to keep costs down, and 30 percent is 30 percent. Besides, these people come highly recommended.

Doug (taking a breath and trying to remain calm): You guys caught me by surprise here, but before you make a final decision, a few comments?

Joe (nodding): Sure, go ahead.

Doug: We haven't worked with this outsourcing company before, right?

Mal: Right, but . . .

Doug: And they note that any changes to spec will be billed at an additional rate, right?

Joe (frowning): True, but we expect that things will be reasonably stable.

Doug: A bad assumption, Joe.

Joe: Well, . . .

Doug: It's likely that we'll release new versions of this product over the next few years. And it's reasonable to assume that software will provide many of the new features, right?

[All nod.]

Doug: Have we ever coordinated an international project before?

Lee (looking concerned): No, but I'm told . . .

Doug (trying to suppress his anger): So, what you're telling me is: (1) we're about to work with an unknown vendor, (2) the costs to do this are not as low as they seem, (3) we're de facto committing to work with them over many product releases, no matter what they do on the first one, and (4) we're going to learn on-the-job relative to an international project.

[All remain silent.]

Doug: Guys . . . I think this is a mistake, and I'd like you to take a day to reconsider. We'll have far more control if we do the work in house. We have the expertise, and I can guarantee that it won't cost us much more . . . the risk will be lower, and I know you're all risk averse, as I am.

Joe (frowning): You've made a few good points, but you have a vested interest in keeping this project in-house.

Doug: That's true, but it doesn't change the facts.

Joe (with a sigh): Okay, let's table this for a day or two, give it some more thought, and meet again for a final decision. Doug, can I speak with you privately?

Doug: Sure . . . I really do want to be sure we do the right thing.

23.11 Summary

The software project planner must estimate three things before a project begins: how long it will take, how much effort will be required, and how many people will be involved. In addition, the planner must predict the resources (hardware and software) that will be required and the risk involved.

The statement of scope helps the planner develop estimates using one or more techniques that fall into two broad categories: decomposition and empirical modeling.

Decomposition techniques require a delineation of major software functions, followed by estimates of either (1) the number of LOC, (2) selected values within the information domain, (3) the number of use-cases, (4) the number of person-months required to implement each function, or (5) the number of person-months required for each software engineering activity. Empirical techniques use empirically derived expressions for effort and time to predict these project quantities. Automated tools can be used to implement a specific empirical model.

Accurate project estimates generally use at least two of the three techniques just noted. By comparing and reconciling estimates derived using different techniques, the planner is more likely to derive an accurate estimate. Software project estimation can never be an exact science, but a combination of good historical data and systematic techniques can improve estimation accuracy.

REFERENCES

[BEN92] Bennatan, E. M., *Software Project Management: A Practitioner's Approach,* McGraw-Hill, 1992.

[BEN03] Bennatan, E. M., ""So What Is the State of Software Estimation?" *The Cutter Edge* (an online newsletter), February 11, 2003, available from http:// www.cutter.com.

[BOE81] Boehm, B., *Software Engineering Economics,* Prentice-Hall, 1981.

[BOE89] Boehm, B., *Risk Management,* IEEE Computer Society Press, 1989.

[BOE96] Boehm, B., "Anchoring the Software Process," IEEE *Software,* vol. 13, no. 4, July 1996, pp. 73–82.

[BOE00] Boehm, B., et al., *Software Cost Estimation in COCOMO II,* Prentice-Hall, 2000.

[BRO75] Brooks, F., *The Mythical Man-Month,* Addison-Wesley, 1975.

[GAU89] Gause, D. C., and G. M. Weinberg, *Exploring Requirements: Quality Before Design,* Dorset House, 1989.

[HOO91] Hooper, J., and R. O. Chester, *Software Reuse: Guidelines and Methods,* Plenum Press, 1991.

[JON96] Jones, C., "How Software Estimation Tools Work," *American Programmer,* vol. 9, no. 7, July 1996, pp. 19–27.

[LOR94] Lorenz, M., and J. Kidd, *Object-Oriented Software Metrics,* Prentice-Hall, 1994.

[MAT94] Matson, J., B. Barrett, and J. Mellichamp, "Software Development Cost Estimation Using Function Points," *IEEE Trans. Software Engineering,* vol. SE-20, no. 4, April 1994, pp. 275–287.

[MCC98] McConnell, S., *Software Project Survival Guide,* Microsoft Press, 1998.

[MEN01] Mendes, E., N. Mosley, and S. Counsell, "Web Metrics—Estimating Design and Authoring Effort," *IEEE Multimedia,* January-March 2001, pp. 50–57.

[MIN95] Minoli, D., *Analyzing Outsourcing,* McGraw-Hill, 1995.

[PHI98] Phillips, D., *The Software Project Manager's Handbook,* IEEE Computer Society Press, 1998.

[PUT78] Putnam, L., "A General Empirical Solution to the Macro Software Sizing and Estimation Problem," *IEEE Trans. Software Engineering,* Vol SE-4, No. 4, July 1978, pp. 345–361.

[PUT92] Putnam, L., and W. Myers, *Measures for Excellence,* Yourdon Press, 1992.

[PUT97a] Putnam, L., and W. Myers, "How Solved Is the Cost Estimation Problem?" *IEEE Software,* November 1997, pp. 105–107.

[PUT97b] Putnam, L., and W. Myers, *Industrial Strength Software: Effective Management Using Measurement,* IEEE Computer Society Press, 1997.

[ROE00] Roetzheim, W., "Estimating Internet Development," *Software Development,* August 2000, available at http://www.sdmagazine.com/documents/s=741/ sdm0008d/0008d.htm.

[SMI99] Smith, J., "The Estimation of Effort Based on Use Cases," Rational Software Corp., 1999, download from http://www.rational.com/media/whitepapers/ finalTP171.PDF.

Problems and Points to Ponder

23.1. Performance is an important consideration during planning. Discuss how performance can be interpreted differently depending upon the software application area.

23.2. Assume that you are the project manager for a company that builds software for household robots. You have been contracted to build the software for a robot that mows the lawn for a homeowner. Write a statement of scope that describes the software. Be sure your statement of scope is bounded. If you're unfamiliar with robots, do a bit of research before you begin writing. Also, state your assumptions about the hardware that will be required. Alternate: Replace the lawn mowing robot with another robotics problem that is of interest to you.

23.3. Software project complexity influences estimation accuracy. Develop a list of software characteristics (e.g., concurrent operation, graphical output) that affect the complexity of a project. Prioritize the list.

23.4. Do a functional decomposition of the robot software you described in Problem 23.2. Estimate the size of each function in LOC. Assuming that your organization produces 450 LOC/pm with a burdened labor rate of $7,000 per person-month, estimate the effort and cost required to build the software using the LOC-based estimation technique described in this chapter.

23.5. Use the COCOMO II model to estimate the effort required to build software for a simple ATM that produces 12 screens, 10 reports, and will require approximately 80 software components. Assume average complexity and average developer/environment maturity. Use the application composition model with object points.

23.6. It seems odd that cost and schedule estimates are developed during software project planning—before detailed software requirements analysis or design has been conducted. Why do you think this is done? Are there circumstances when it should not be done?

23.7. Use the software equation to estimate the lawn mowing robot software from Problem 23.2. Assume that Equations (23-5) are applicable and that $P = 8000$.

23.8. Compare the effort estimates derived in Problems 23.4 and 23.7. What is the standard deviation, and how does it affect your degree of certainty about the estimate?

23.9. Using the results obtained in Problem 23.8, determine whether it's reasonable to expect that the software can be built within the next six months and how many people would have to be used to get the job done.

23.11. Develop a spreadsheet model that implements one or more of the estimation techniques described in this chapter. Alternatively, acquire one or more on-line models for software project estimation from Web-based sources.

23.10. For a project team, develop a software tool that implements each of the estimation techniques developed in this chapter.

23.12. Recompute the expected values noted for the decision tree in Figure 23.8 assuming that every branch has a 50–50 probability. Would this change your final decision?

Further Readings and Information Sources

Most software project management books contain discussions of project estimation. The Project Management Institute (*PMBOK Guide,* PMI, 2001), Wysoki and his colleagues (*Effective Project Management,* Wiley, 2000), Lewis (*Project Planning Scheduling and Control,* third edition, McGraw-Hill, 2000), Bennatan (*On Time, Within Budget: Software Project Management Practices and Techniques,* third edition, Wiley, 2000), and Phillips [PHI98] provide useful estimation guidelines.

Jones (*Estimating Software Costs,* McGraw-Hill, 1998) has written one of the most comprehensive treatments of the subject published to date. His book contains models and data that are applicable to software estimating in every application domain. Coombs (*IT Project Estimation,* Cambridge University Press, 2002), Roetzheim and Beasley (*Software Project Cost and Schedule Estimating: Best Practices,* Prentice-Hall, 1997), and Wellman (*Software Costing,* Prentice-Hall, 1992) present many useful models and suggest step-by-step guidelines for generating the best possible estimates.

Putnam and Myer's detailed treatment of software cost estimating ([PUT92] and [PUT97b]) and Boehm's books on software engineering economics ([BOE81] and COCOMO II [BOE00]) describe empirical estimation models. These books provide detailed analysis of data derived from hundreds of software projects. An excellent book by DeMarco (*Controlling Software Projects,* Yourdon Press, 1982) provides valuable insight into the management, measurement, and estimation of software projects. Lorenz and Kidd (*Object-Oriented Software Metrics,* Prentice-Hall, 1994) and Cockburn (*Surviving Object-Oriented Projects,* Addison-Wesley, 1998) consider estimation for object-oriented systems.

A wide variety of information sources on software estimation is available on the Internet. An up-to-date list of World Wide Web references can be found at the SEPA Web site:
http://www.mhhe.com/pressman.

[illegible]

Chapter

Project Scheduling 24

Key Concepts
basic principles
lateness
earned value
effort distribution
people and effort
PNR curve
task network
task refinement
time-boxing
timeline charts
tracking
work breakdown

In the late 1960s, a bright-eyed young engineer was chosen to "write" a computer program for an automated manufacturing application. The reason for his selection was simple. He was the only person in his technical group who had attended a computer programming seminar. He knew the ins and outs of assembly language and FORTRAN but nothing about software engineering and even less about project scheduling and tracking.

His boss gave him the appropriate manuals and a verbal description of what had to be done. He was informed that the project must be completed in two months.

He read the manuals, considered his approach, and began writing code. After two weeks, the boss called him into his office and asked how things were going.

"Really great," said the young engineer with youthful enthusiasm, "This was much simpler than I thought. I'm probably close to 75 percent finished."

The boss smiled and encouraged the young engineer to keep up the good work. They planned to meet again in a week's time.

A week later the boss called the engineer into his office and asked, "Where are we?"

"Everything's going well," said the youngster, "but I've run into a few small snags. I'll get them ironed out and be back on track soon."

"How does the deadline look?" the boss asked.

Quick Look

What is it? You've selected an [illegible] estimated the amount of [illegible] people, you know [illegible] the risks. Now it's time to [illegible] you have to [illegible] engineering tasks [illegible] done on time. Once [illegible] to assign responsibility [illegible] gets done, and adjust the [illegible] reality. In a nutshell, that's software project scheduling and tracking.

Who does it? At the project level, [illegible] ect managers using information [illegible] software engineers. At an individual level, software engineers themselves.

Why is it important? To build a complex system, many software engineering tasks occur in [illegible] and the result of work performed during [illegible] may have a profound effect on [illegible] another task. These interde- [illegible] difficult to understand with- [illegible] impossible to [illegible] software [illegible] engineer- [illegible] process [illegible] be [illegible] each [illegible]

What is the work product? The project schedule and related information are produced.
How do I ensure that I've done it right? Proper scheduling requires that (1) all tasks appear in the network, (2) effort and timing are intelligently allocated to each task, (3) interdependencies between tasks are properly indicated, (4) resources are allocated for the work to be done, and (5) closely spaced milestones are provided so that progress can be tracked.

"No problem," said the engineer. "I'm close to 90 percent complete."

If you've been working in the software world for more than a few years, you can finish the story. It'll come as no surprise that the young engineer[1] stayed 90 percent complete for the entire project duration and finished (with the help of others) only one month late.

This story has been repeated tens of thousands of times by software developers during the past four decades. The big question is why?

24.1 Basic Concepts

Although there are many reasons why software is delivered late, most can be traced to one or more of the following root causes:

- An unrealistic deadline established by someone outside the software engineering group and forced on managers and practitioners within the group.
- Changing customer requirements that are not reflected in schedule changes.
- An honest underestimate of the amount of effort and/or the number of resources that will be required to do the job.
- Predictable and/or unpredictable risks that were not considered when the project commenced.
- Technical difficulties that could not have been foreseen in advance.
- Human difficulties that could not have been foreseen in advance.
- Miscommunication among project staff that results in delays.
- A failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem.

"Unrealistic or irrational schedules are probably the single most destructive influence in all of software."
Capers Jones

Aggressive (read "unrealistic") deadlines are a fact of life in the software business. Sometimes such deadlines are demanded for reasons that are legitimate, from the point of view of the person who sets the deadline. But common sense says that legitimacy must also be perceived by the people doing the work.

1 In case you were wondering, this story is autobiographical.

Napoleon once said: "Any commander-in-chief who undertakes to carry out a plan which he considers defective is at fault; he must put forth his reasons, insist on the plan being changed, and finally tender his resignation rather than be the instrument of his army's downfall." These are strong words that many software project managers should ponder.

The estimation activities discussed in Chapter 23 and the scheduling techniques described in this chapter are often implemented under the constraint of a defined deadline. If best estimates indicate that the deadline is unrealistic, a competent project manager should "protect his or her team from undue [schedule] pressure . . . [and] reflect the pressure back to its originators" [PAG85].

To illustrate, assume that a software engineering team has been asked to build a real-time controller for a medical diagnostic instrument that is to be introduced to the market in nine months. After careful estimation and risk analysis (Chapter 25), the software project manager comes to the conclusion that the software, as requested, will require 14 calendar months to create with available staff. How does the project manager proceed?

"I love deadlines. I like the whooshing sound they make as they fly by."

Douglas Adams

It is unrealistic to march into the customer's office (in this case the likely customer is marketing/sales) and demand that the delivery date be changed. External market pressures have dictated the date, and the product must be released. It is equally foolhardy to refuse to undertake the work (from a career standpoint). So, what to do? The following steps are recommended in this situation:

? What should we do when management demands that we make a deadline that is impossible?

1. Perform a detailed estimate using historical data from past projects. Determine the estimated effort and duration for the project.
2. Using an incremental process model (Chapter 3), develop a software engineering strategy that will deliver critical functionality by the imposed deadline, but delay other functionality until later. Document the plan.
3. Meet with the customer and (using the detailed estimate), explain why the imposed deadline is unrealistic. Be certain to note that all estimates are based on performance on past projects. Also be certain to indicate the percent improvement that would be required to achieve the deadline as it currently exists.[2] The following comment is appropriate:

   "I think we may have a problem with the delivery date for the XYZ controller software. I've given each of you an abbreviated breakdown of production rates

2 If the required improvement is 10 to 25 percent, it may actually be possible to get the job done. But, more likely, the required improvement in team performance will be greater than 50 percent. This is an unrealistic expectation.

for past projects and an estimate that we've done a number of different ways. You'll note that I've assumed a 20 percent improvement in past production rates, but we still get a delivery date that's 14 calendar months rather than 9 months away."

4. Offer the incremental development strategy as an alternative:

   "We have a few options, and I'd like you to make a decision based on them. First, we can increase the budget and bring on additional resources so that we'll have a shot at getting this job done in nine months. But understand that this will increase risk of poor quality due to the tight timeline.[3] Second, we can remove a number of the software functions and capabilities that you're requesting. This will make the preliminary version of the product somewhat less functional, but we can announce all functionality and then deliver over the 14 month period. Third, we can dispense with reality and wish the project complete in nine months. We'll wind up with nothing that can be delivered to a customer. The third option, I hope you'll agree, is unacceptable. Past history and our best estimates say that it is unrealistic and a recipe for disaster."

There will be some grumbling, but if solid estimates based on good historical data are presented, it's likely that negotiated versions of option 1 or 2 will be chosen. The unrealistic deadline evaporates.

24.2 PROJECT SCHEDULING

Fred Brooks, the well-known author of *The Mythical Man-Month* [BRO95], was once asked how software projects fall behind schedule. His response was as simple as it was profound: "One day at a time."

The reality of a technical project (whether it involves building a hydroelectric plant or developing an operating system) is that hundreds of small tasks must occur to accomplish a larger goal. Some of these tasks lie outside the mainstream and may be completed without worry about impact on project completion date. Other tasks lie on the "critical path." If these "critical" tasks fall behind schedule, the completion date of the entire project is put into jeopardy.

ADVICE

The tasks required to achieve a project manager's objective should not be performed manually. There are many excellent scheduling tools. Use them.

The project manager's objective is to define all project tasks, build a network that depicts their interdependencies, identify the tasks that are critical within the network, and then track their progress to ensure that delay is recognized "one day at a time." To accomplish this, the manager must have a schedule that has been defined at a degree of resolution that allows progress to be monitored and the project to be controlled.

3 You might also add that increasing the number of people does not reduce calendar time proportionally.

Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks. It is important to note, however, that the schedule evolves over time. During early stages of project planning, a macroscopic schedule is developed. This type of schedule identifies all major process framework activities and the product functions to which they are applied. As the project gets under way, each entry on the macroscopic schedule is refined into a detailed schedule. Here, specific software tasks (required to accomplish an activity) are identified and scheduled.

"Overly optimistic scheduling doesn't result in shorter actual schedules, it results in longer ones."
Steve McConnell

Scheduling for software engineering projects can be viewed from two rather different perspectives. In the first, an end-date for release of a computer-based system has already (and irrevocably) been established. The software organization is constrained to distribute effort within the prescribed time frame. The second view of software scheduling assumes that rough chronological bounds have been discussed but that the end-date is set by the software engineering organization. Effort is distributed to make best use of resources and an end-date is defined after careful analysis of the software. Unfortunately, the first situation is encountered far more frequently than the second.

24.2.1 Basic Principles

Like all other areas of software engineering, a number of basic principles guide software project scheduling:

Compartmentalization. The project must be compartmentalized into a number of manageable activities, actions, and tasks. To accomplish compartmentalization, both the product and the process are decomposed.

KEY POINT
When you develop a schedule, compartmentalize the work, note task interdependencies, allocate effort and time to each task, define responsibilities, outcomes, and milestones.

Interdependency. The interdependency of each compartmentalized activity, action, or task must be determined. Some tasks must occur in sequence while others can occur in parallel. Some actions or activities cannot commence until the work product produced by another is available. Other actions or activities can occur independently.

Time allocation. Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort). In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis.

Effort validation. Every project has a defined number of people on the software team. As time allocation occurs, the project manager must ensure that no more than the allocated number of people have been scheduled at any given time. For example, consider a project that has three assigned software engineers (e.g., three person-days are available per day of assigned effort[4]). On a given day, seven concurrent tasks must be accomplished. Each task requires 0.50 person days of effort. More effort has been allocated than there are people to do the work.

Defined responsibilities. Every task that is scheduled should be assigned to a specific team member.

Defined outcomes. Every task that is scheduled should have a defined outcome. For software projects, the outcome is normally a work product (e.g., the design of a module) or a part of a work product. Work products are often combined in deliverables.

Defined milestones. Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality (Chapter 26) and has been approved.

Each of these principles is applied as the project schedule evolves.

24.2.2 The Relationship Between People and Effort

In a small software development project a single person can analyze requirements, perform design, generate code, and conduct tests. As the size of a project increases, more people must become involved. (We can rarely afford the luxury of approaching a 10 person-year effort with one person working for 10 years!)

ADVICE

If you must add people to a late project, be sure that you've assigned them work that is highly compartmentalized.

There is a common myth that is still believed by many managers who are responsible for software development effort: "If we fall behind schedule, we can always add more programmers and catch up later in the project." Unfortunately, adding people late in a project often has a disruptive effect on the project, causing schedules to slip even further. The people who are added must learn the system, and the people who teach them are the same people who were doing the work. During teaching, no work is done, and the project falls further behind.

In addition to the time it takes to learn the system, more people increase the number of communication paths and the complexity of communication throughout a project. Although communication is absolutely essential to successful software development, every new communication path requires additional effort and therefore additional time.

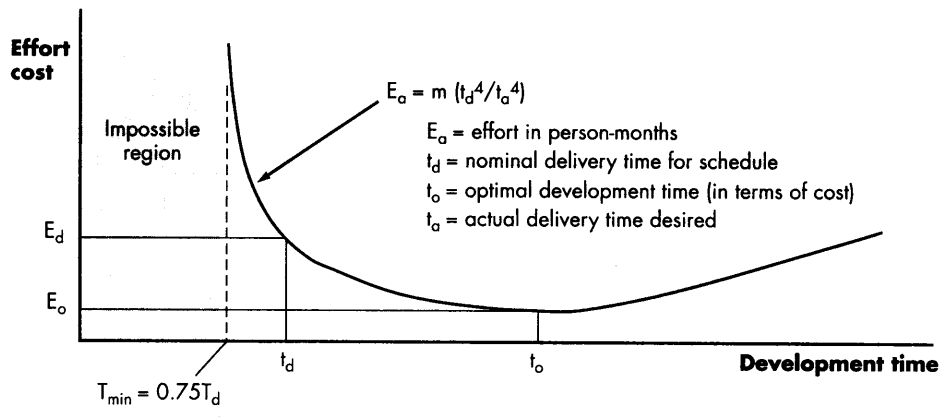
Over the years, empirical data and theoretical analysis have demonstrated that project schedules are elastic. That is, it is possible to compress a desired project completion date (by adding additional resources) to some extent. It is also possible to extend a completion date (by reducing the number of resources).

The *Putnam-Norden-Rayleigh (PNR) Curve*[5] provides an indication of the relationship between effort applied and delivery time for a software project. A version of the curve,

4 In reality, less than three person-days of effort are available because of unrelated meetings, sickness, vacation, and a variety of other reasons. For our purposes, however, we assume 100 percent availability.

5 Original research can be found in [NOR70] and [PUT78].

FIGURE 24.1 **The relationship between effort and delivery time**



representing project effort as a function of delivery time, is shown in Figure 24.1. The curve indicates a minimum value, t_o, that indicates the least cost time for delivery (i.e., the delivery time that will result in the least effort expended). As we move left of t_o (i.e., as we try to accelerate delivery), the curve rises nonlinearly.

KEY POINT

If delivery can be delayed, the PNR curve indicates that project costs can be reduced substantially.

As an example, we assume that a project team has estimated a level of effort, E_a, will be required to achieve a nominal delivery time, t_d, that is optimal in terms of schedule and available resources. Although it is possible to accelerate delivery, the curve rises very sharply to the left of t_d. In fact, the PNR curve indicates that the project delivery time cannot be compressed much beyond 0.75 t_d. If we attempt further compression, the project moves into "the impossible region" and risk of failure becomes very high. The PNR curve also indicates that the lowest cost delivery option, $t_o = 2\ t_d$. The implication here is that delaying project delivery can reduce costs significantly. Of course, this must be weighed against the business cost associated with the delay.

The software equation [PUT92] introduced in Chapter 23 is derived from the PNR curve and demonstrates the highly nonlinear relationship between chronological time to complete a project and human effort applied to the project. The number of delivered lines of code (source statements), L, is related to effort and development time by the equation:

ADVICE

As the project deadline becomes tighter and tighter, you reach a point at which the work cannot be completed on schedule, regardless of the number of people doing the work. Face reality and define a new delivery date.

$$L = P \times E^{1/3}t^{4/3}$$

where E is development effort in person-months, P is a productivity parameter that reflects a variety of factors that lead to high-quality software engineering work (typical values for P range between 2000 and 12,000), and t is the project duration in calendar months.

Rearranging this software equation, we can arrive at an expression for development effort E:

$$E = L^3/(P^3t^4) \qquad (24\text{-}1)$$

where E is the effort expended (in person-years) over the entire life cycle for software development and maintenance, and t is the development time in years. The equation for development effort can be related to development cost by the inclusion of a burdened labor rate factor ($/person-year).

This leads to some interesting results. Consider a complex, real-time software project estimated at 33,000 LOC, 12 person-years of effort. If eight people are assigned to the project team, the project can be completed in approximately 1.3 years. If, however, we extend the end-date to 1.75 years, the highly nonlinear nature of the model described in Equation (24-1) yields:

$$E = L^3/(P^3t^4) \sim 3.8 \text{ person-years.}$$

This implies that, by extending the end-date six months, we can reduce the number of people from eight to four! The validity of such results is open to debate, but the implication is clear: benefit can be gained by using fewer people over a somewhat longer time span to accomplish the same objective.

24.2.3 Effort Distribution

How should effort be distributed across the software process workflow?

Each of the software project estimation techniques discussed in Chapter 23 leads to estimates of work units (e.g., person-months) required to complete software development. A recommended distribution of effort across the software process is often referred to as the *40–20–40 rule*. Forty percent of all effort is allocated to front-end analysis and design. A similar percentage is applied to back-end testing. You can correctly infer that coding (20 percent of effort) is deemphasized.

This effort distribution should be used as a guideline only.[6] The characteristics of each project must dictate the distribution of effort. Work expended on project planning rarely accounts for more than 2–3 percent of effort, unless the plan commits an organization to large expenditures with high risk. Requirements analysis may comprise 10–25 percent of project effort. Effort expended on analysis or prototyping should increase in direct proportion with project size and complexity. A range of 20 to 25 percent of effort is normally applied to software design. Time expended for design review and subsequent iteration must also be considered.

Because of the effort applied to software design, code should follow with relatively little difficulty. A range of 15–20 percent of overall effort can be achieved. Testing and subsequent debugging can account for 30–40 percent of software development effort. The criticality of the software often dictates the amount of testing that is required. If software is human rated (i.e., software failure can result in loss of life), even higher percentages are typical.

6 Today, the 40-20-40 rule is under attack. Some believe that more than 40 percent of overall effort should be expended during analysis and design. On the other hand, some proponents of agile development (Chapter 4) argue that less time should be expended "up front" and that a team should move quickly to construction.

24.3 Defining a Task Set for the Software Project

A number of different process models were described in Part 1 of this book. Regardless of whether a software team chooses a linear sequential model, an incremental model, an evolutionary model, or some permutation, the process model is populated by a set of tasks that enables a software team to define, develop, and ultimately support computer software.

No single task set is appropriate for all projects. The set of tasks that would be appropriate for a large, complex system would likely be perceived as overkill for a small, relatively simple software product. Therefore, an effective software process should define a collection of task sets, each designed to meet the needs of different types of projects.

As we noted in Chapter 2, a task set is a collection of software engineering work tasks, milestones, and work products that must be accomplished to complete a particular project. The task set should provide enough discipline to achieve high software quality. But, at the same time, it must not burden the project team with unnecessary work.

To develop a project schedule, a task set must be distributed on the project time line. The task set will vary depending upon the project type and the degree of rigor with which the software team decides to do its work. Although it is difficult to develop a comprehensive taxonomy of software project types, most software organizations encounter the following projects:

1. *Concept development projects* that are initiated to explore some new business concept or application of some new technology.
2. *New application development* projects that are undertaken as a consequence of a specific customer request.
3. *Application enhancement* projects that occur when existing software undergoes major modifications to function, performance, or interfaces that are observable by the end-user.
4. *Application maintenance projects* that correct, adapt, or extend existing software in ways that may not be immediately obvious to the end-user.
5. *Reengineering projects* that are undertaken with the intent of rebuilding an existing (legacy) system in whole or in part.

WebRef
An adaptable process model (APM) has been developed to assist in defining task sets for various software projects. A complete description of the APM can be found at www.rspa.com/apm.

Even within a single project type, many factors influence the task set to be chosen. These include [PRE99]: size of the project, number of potential users, mission criticality, application longevity, stability of requirements, ease of customer/developer communication, maturity of applicable technology, performance constraints, embedded and nonembedded characteristics, project staff, and reengineering factors. When taken in combination, these factors provide an indication of the *degree of rigor* with which the software process should be applied.

24.3.1 A Task Set Example

Each of the project types described may be approached using a process model that is linear sequential, iterative (e.g., the prototyping or incremental models), or evolutionary (e.g., the spiral model). In some cases, one project type flows smoothly into the next. For example, concept development projects that succeed often evolve into new application development projects. As a new application development project ends, an application enhancement project sometimes begins. This progression is both natural and predictable and will occur regardless of the process model that is adopted by an organization. Therefore, the major software engineering tasks described in the sections that follows are applicable to all process model flows. As an example, we consider the software engineering tasks for a concept development project.

Concept development projects are initiated when the potential for some new technology must be explored. There is no certainty that the technology will be applicable, but a customer (e.g., marketing) believes that potential benefit exists. Concept development projects are approached by applying the following major tasks:

1.1 **Concept scoping** determines the overall scope of the project.

1.2 **Preliminary concept planning** establishes the organization's ability to undertake the work implied by the project scope.

1.3 **Technology risk assessment** evaluates the risk associated with the technology to be implemented as part of project scope.

1.4 **Proof of concept** demonstrates the viability of a new technology in the software context.

1.5 **Concept implementation** implements the concept representation in a manner that can be reviewed by a customer and is used for "marketing" purposes when a concept must be sold to other customers or management.

1.6 **Customer reaction** to the concept solicits feedback on a new technology concept and targets specific customer applications.

A quick scan of these tasks should yield few surprises. In fact, the software engineering flow for concept development projects (and for all other types of projects as well) is little more than common sense.

24.3.2 Refinement of Major Tasks

The major tasks described in the preceding section may be used to define a macroscopic schedule for a project. However, the macroscopic schedule must be refined to create a detailed project schedule. Refinement begins by taking each major task and decomposing it into a set of subtasks (with related work products and milestones).

As an example of task decomposition, consider Task 1.1, Concept Scoping. Task refinement can be accomplished using an outline format, but in this book, a process design language approach is used to illustrate the flow of the concept scoping activity:

```
Task definition: Task 1.1 Concept Scoping
1.1.1 Identify need, benefits and potential customers;
1.1.2 Define desired output/control and input events that drive the application;
      Begin Task 1.1.2
      1.1.2.1 FTR: Review written description of need[7]
      1.1.2.2 Derive a list of customer visible outputs/inputs
      1.1.2.3 FTR: Review outputs/inputs with customer and revise as required;
      endtask Task 1.1.2
1.1.3 Define the functionality/behavior for each major function;
      Begin Task 1.1.3
      1.1.3.1 FTR: Review output and input data objects derived in task 1.1.2;
      1.1.3.2 Derive a model of functions/behaviors;
      1.1.3.3 FTR: Review functions/behaviors with customer and revise as required;
      endtask Task 1.1.3
1.1.4 Isolate those elements of the technology to be implemented in software;
1.1.5 Research availability of existing software;
1.1.6 Define technical feasibility;
1.1.7 Make quick estimate of size;
1.1.8 Create a Scope Definition;
endTask definition: Task 1.1
```

The tasks and subtasks noted in the process design language refinement form the basis for a detailed schedule for the concept scoping activity.

24.4 Defining a Task Network

KEY POINT
The task network is a useful mechanism for depicting intertask dependencies and determining the critical path.

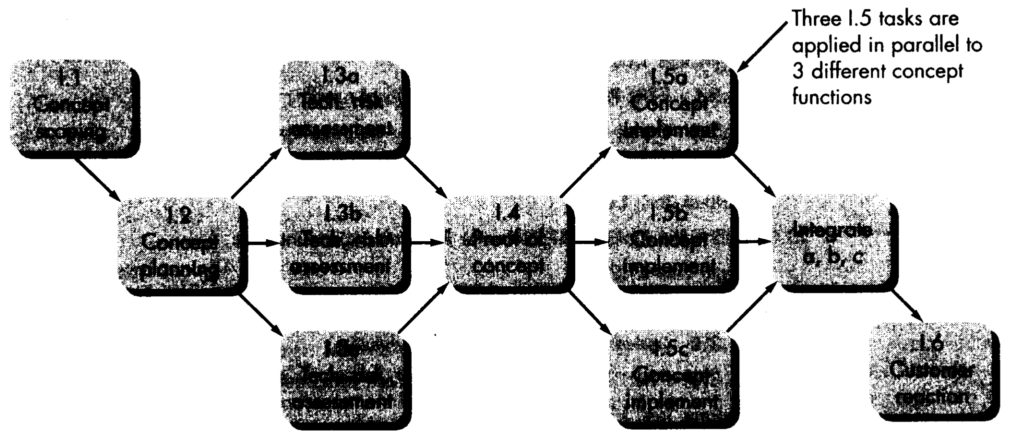
Individual tasks and subtasks have interdependencies based on their sequence. In addition, when more than one person is involved in a software engineering project, it is likely that development activities and tasks will be performed in parallel. When this occurs, concurrent tasks must be coordinated so that they will be complete when later tasks require their work product(s).

A *task network*, also called an *activity network*, is a graphic representation of the task flow for a project. It is sometimes used as the mechanism through which task sequence and dependencies are input to an automated project scheduling tool. In its simplest form (used when creating a macroscopic schedule), the task network depicts major software engineering tasks. Figure 24.2 shows a schematic task network for a concept development project.

The concurrent nature of software engineering activities leads to a number of important scheduling requirements. Because parallel tasks occur asynchronously, the planner must determine intertask dependencies to ensure continuous progress

7 FTR indicates that a formal technical review (Chapter 26) is to be conducted.

FIGURE 24.2 A task network for concept development



toward completion. In addition, the project manager should be aware of those tasks that lie on the *critical path*. That is, tasks that must be completed on schedule if the project as a whole is to be completed on schedule. These issues are discussed in more detail later in this chapter.

It is important to note that the task network shown in Figure 24.2 is macroscopic. In a detailed task network (a precursor to a detailed schedule), each activity shown in the figure would be expanded. For example, Task 1.1 would be expanded to show all tasks detailed in the refinement of Tasks 1.1 shown in Section 24.3.2.

[illegible]

Scheduling of a software project does not differ greatly from scheduling of any multitask engineering effort. Therefore, generalized project scheduling tools and techniques can be applied with little modification for software projects.

Program evaluation and review technique (PERT) and the *critical path method* (CPM) are two project scheduling methods that can be applied to software development. Both techniques are driven by information already developed in earlier project planning activities:

- Estimates of effort.
- A decomposition of the product function.
- The selection of the appropriate process model and task set.
- Decomposition of tasks.

Interdependencies among tasks may be defined using a task network. Tasks, sometimes called the project *work breakdown structure* (WBS), are defined for the product as a whole or for individual functions.

"All we have to decide is what to do with the time that is given to us."
Gandalf in *The Lord of the Rings: Fellow* [illegible]

Both PERT and CPM provide quantitative tools that allow the software planner to (1) determine the critical path—the chain of tasks that determines the duration of the project; (2) establish "most likely" time estimates for individual tasks by applying statistical models; and (3) calculate "boundary times" that define a time "window" for a particular task.

SOFTWARE TOOLS

Project Scheduling

Objective: The objective of project scheduling tools is to enable a project manager to define work tasks, establish their dependencies, assign human resources to tasks, and develop a variety of graphs, charts, and tables that aid in tracking and control of the software project.

Mechanics: In general, project scheduling tools require the specification of a work breakdown structure or the generation of a task network. Once the task breakdown (an outline) or network is defined, start and end dates, human resources, hard deadlines, and other data are attached to each task. The tool then generates a variety of timeline charts and other tables that enable a manager to assess the task flow of a project. These data can be updated continually as the project is conducted.

Representative Tools[8]

AMS Realtime, developed by Advanced Management Systems (www.amsusa.com), provides scheduling capabilities for projects of all sizes and types.

Microsoft Project, developed by Microsoft (www.microsoft.com), is the most widely used PC-based project scheduling tool.

Viewpoint, developed by Artemis Internation Solutions Corp. (www.atemispm.com), supports all aspects of project planning including scheduling.

A comprehensive list of project management software vendors and products can be found at www.infogoal.com/pmc/pmcswr.htm.

24.5.1 Timeline Charts

When creating a software project schedule, the planner begins with a set of tasks (the work breakdown structure). If automated tools are used, the work breakdown is input as a task network or task outline. Effort, duration, and start date are then input for each task. In addition, tasks may be assigned to specific individuals.

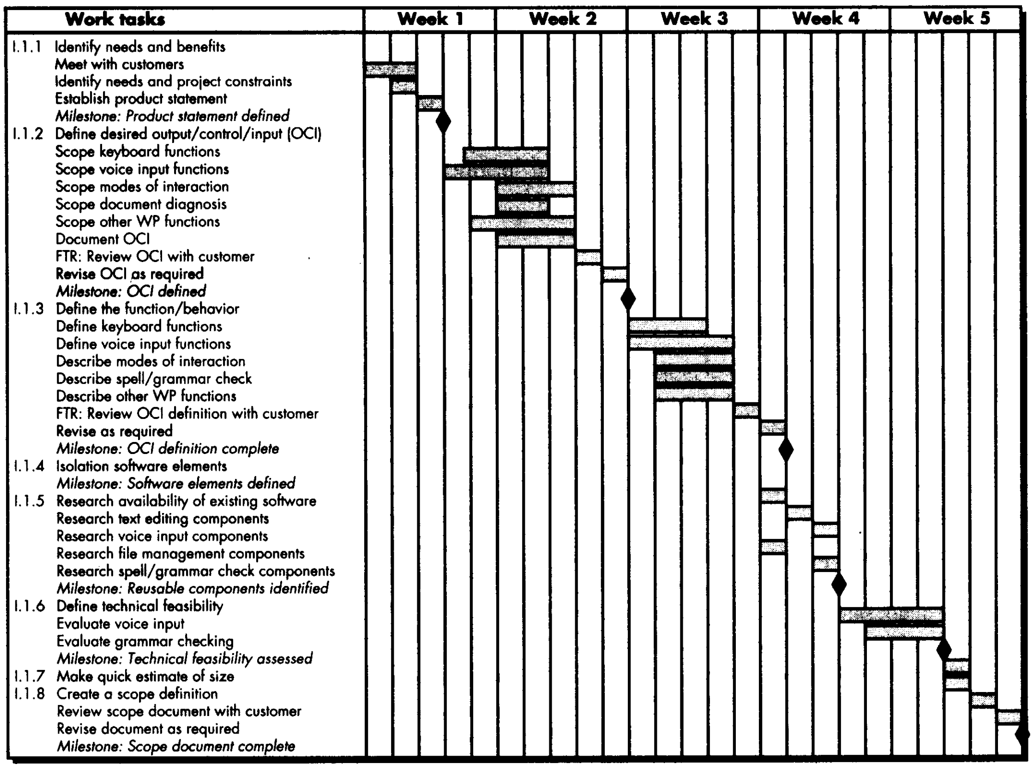
KEY POINT
A timeline chart enables you to determine what tasks will be conducted at a given point in time.

As a consequence of this input, a *timeline chart,* also called a *Gantt chart,* is generated. A timeline chart can be developed for the entire project. Alternatively, separate charts can be developed for each project function or for each individual working on the project.

Figure 24.3 illustrates the format of a timeline chart. It depicts a part of a software project schedule that emphasizes the concept scoping task for a word-processing (WP) software product. All project tasks (for concept scoping) are listed in

8 Tools noted here do not represent an endorsement, but rather a sampling of tools in this category. In most cases, tool names are trademarked by their respective developers.

FIGURE 24.3 **An example timeline chart**



the left-hand column. The horizontal bars indicate the duration of each task. When multiple bars occur at the same time on the calendar, task concurrency is implied. The diamonds indicate milestones.

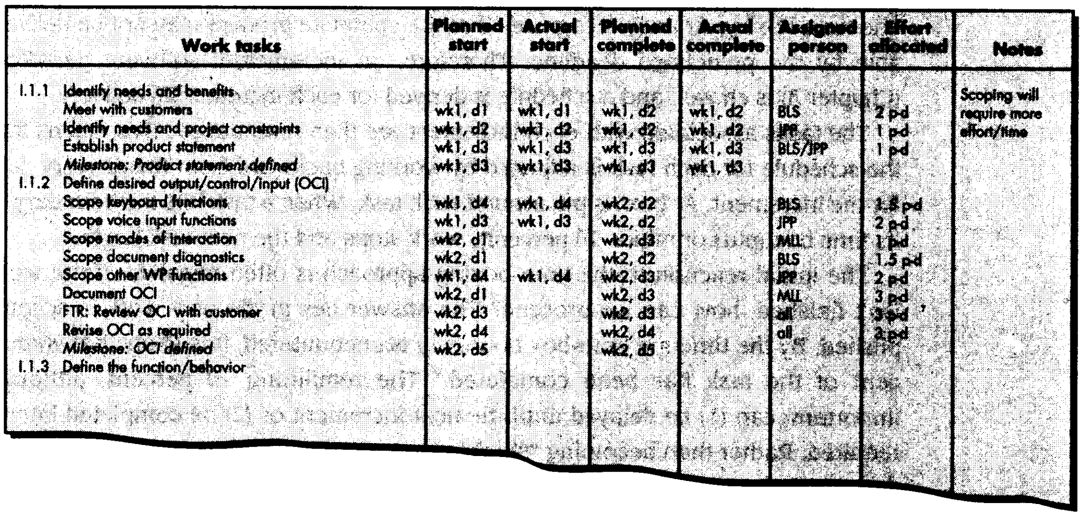
Once the information necessary for the generation of a timeline chart has been input, the majority of software project scheduling tools produce *project tables*—a tabular listing of all project tasks, their planned and actual start- and end-dates, and a variety of related information (Figure 24.4). Used in conjunction with the timeline chart, project tables enable the project manager to track progress.

24.5.2 Tracking the Schedule

The project schedule provides a road map for a software project manager. If it has been properly developed, the project schedule defines the tasks and milestones that must be tracked and controlled as the project proceeds. Tracking can be accomplished in a number of different ways:

- Conducting periodic project status meetings in which each team member reports progress and problems.

FIGURE 24.4 An example resource table



Work tasks	Planned start	Actual start	Planned complete	Actual complete	Assigned person	Effort allocated	Notes
I.1.1 Identify needs and benefits							Scoping will require more effort/time
Meet with customers	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 p-d	
Identify needs and project constraints	wk1, d2	wk1, d2	wk1, d2	wk1, d2	JPP	1 p-d	
Establish product statement	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 p-d	
Milestone: Product statement defined	wk1, d3	wk1, d3	wk1, d3	wk1, d3			
I.1.2 Define desired output/control/input (OCI)							
Scope keyboard functions	wk1, d4	wk1, d4	wk2, d2		BLS	1.5 p-d	
Scope voice input functions	wk1, d3	wk1, d3	wk2, d2		JPP	2 p-d	
Scope modes of interaction	wk2, d1		wk2, d3		MLL	1 p-d	
Scope document diagnostics	wk2, d1		wk2, d2		BLS	1.5 p-d	
Scope other WP functions	wk1, d4	wk1, d4	wk2, d3		JPP	2 p-d	
Document OCI	wk2, d1		wk2, d3		MLL	3 p-d	
FTR: Review OCI with customer	wk2, d3		wk2, d3		all	3 p-d	
Revise OCI as required	wk2, d4		wk2, d4		all	3 p-d	
Milestone: OCI defined	wk2, d5		wk2, d5				
I.1.3 Define the function/behavior							

- Evaluating the results of all reviews conducted throughout the software engineering process.
- Determining whether formal project milestones (the diamonds shown in Figure 24.3) have been accomplished by the scheduled date.
- Comparing actual start-date to planned start-date for each project task listed in the resource table (Figure 24.4).
- Meeting informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon.
- Using earned value analysis (Section 24.6) to assess progress quantitatively.

In reality, all of these tracking techniques are used by experienced project managers.

"The basic rule of software status reporting can be summarized in a single phrase: No surprises."

Capers Jones

ADVICE

The best indication of progress is the completion and successful review of a defined software work product.

Control is employed by a software project manager to administer project resources, cope with problems, and direct project staff. If things are going well (i.e., the project is on schedule and within budget, reviews indicate that real progress is being made, and milestones are being reached), control is light. But when problems occur, the project manager must exercise control to reconcile them as quickly as possible. After a problem has been diagnosed, additional resources may be focused on the problem area: staff may be redeployed or the project schedule can be redefined.

When faced with severe deadline pressure, experienced project managers sometimes use a project scheduling and control technique called *time-boxing* [ZAH95]. The time-boxing strategy recognizes that the complete product may not be deliverable by the predefined deadline. Therefore, an incremental software paradigm (Chapter 3) is chosen and a schedule is derived for each incremental delivery.

The tasks associated with each increment are then time-boxed. This means that the schedule for each task is adjusted by working backward from the delivery date for the increment. A "box" is put around each task. When a task hits the boundary of its time box (plus or minus 10 percent), work stops and the next task begins.

The initial reaction to the time-boxing approach is often negative: If the work isn't finished, how can we proceed? The answer lies in the way work is accomplished. By the time the time-box boundary is encountered, it is likely that 90 percent of the task has been completed.[9] The remaining 10 percent, although important, can (1) be delayed until the next increment or (2) be completed later if required. Rather than becoming "stuck" on a task, the project proceeds toward the delivery date.

24.5.3 Tracking Progress for an OO Project

Although an iterative model is the best framework for an OO project, task parallelism makes project tracking difficult. The project manager can have difficulty establishing meaningful milestones for an OO project because a number of different things are happening at once. In general, the following major milestones can be considered "completed" when the criteria noted have been met.

Technical milestone: OO analysis completed

- All classes and the class hierarchy have been defined and reviewed.
- Class attributes and operations associated with a class have been defined and reviewed.
- Class relationships (Chapter 8) have been established and reviewed.
- A behavioral model (Chapter 8) has been created and reviewed.
- Reusable classes have been noted.

Technical milestone: OO design completed

- The set of subsystems (Chapter 9) has been defined and reviewed.
- Classes are allocated to subsystems and reviewed.
- Task allocation has been established and reviewed.
- Responsibilities and collaborations (Chapters 8 and 9) have been identified.

9 A cynic might recall the saying: The first 90 percent of the system takes 90 percent of the time; the remaining 10 percent of the system takes 90 percent of the time.

- Design classes have been created and reviewed.
- The communication model has been created and reviewed.

Technical milestone: OO programming completed

- Each new class has been implemented in code from the design model.
- Extracted classes (from a reuse library) have been implemented.
- Prototype or increment has been built.

Technical milestone: OO testing

- The correctness and completeness of OO analysis and design models has been reviewed.
- A class-responsibility-collaboration network (Chapter 8) has been developed and reviewed.
- Test cases are designed, and class-level tests (Chapter 14) have been conducted for each class.
- Test cases are designed, and cluster testing (Chapter 14) is completed and the classes are integrated.
- System level tests have been completed.

ADVICE

Debugging and testing occur in concert with one another. The status of debugging is often assessed by considering the type and number of "open" errors (bugs).

Recalling that the OO process model is iterative, each of these milestones may be revisited as different increments are delivered to the customer.

SAFEHOME

[illegible] the Schedule

[illegible] Doug Miller's office, prior [illegible] software project.

[illegible] (manager of the *SafeHome* [illegible] and Vinod Raman, Jamie [illegible] of the product software

[illegible]

[illegible] Powerpoint slide): The [illegible] increment seems [illegible] going to have trouble tracking [illegible]

[illegible] look on his face): Why? We [illegible] a daily basis, plenty of work [illegible] sure that we're not over-[illegible]

[illegible] how do we know when the analysis [illegible] increment is complete?

Jamie: Things are iterative, [illegible]

Doug: I understand that, but [illegible] *analysis classes defined.* [illegible] milestone.

Vinod: We have.

Doug: Who makes that [illegible]

Jamie (aggravated): [illegible] done.

Doug: That's not good [illegible] schedule FTRs [formal [illegible] and you haven't done that. [illegible] review on the analysis [illegible] reasonable milestone. [illegible]

Jamie (frowning): [illegible]

Doug: It shouldn't take [illegible] corrections . . . everyone [illegible]

24.6 Earned Value Analysis

KEY POINT
Earned value provides a quantitative indication of progress.

In Section 24.5, we discussed a number of qualitative approaches to project tracking. Each provides the project manager with an indication of progress, but an assessment of the information provided is somewhat subjective. It is reasonable to ask whether there is a quantitative technique for assessing progress as the software team moves through the work tasks allocated to the project schedule. In fact, a technique for performing quantitative analysis of progress does exist. It is called *earned value analysis* (EVA). Humphrey [HUM95] discusses earned value in the following manner:

> The earned value system provides a common value scale for every [software project] task, regardless of the type of work being performed. The total hours to do the whole project are estimated, and every task is given an earned value based on its estimated percentage of the total.

Stated even more simply, earned value is a measure of progress. It enables us to assess the "percent of completeness" of a project using quantitative analysis rather than rely on a gut feeling. In fact, Fleming and Koppleman [FLE98] argue that earned value analysis "provides accurate and reliable readings of performance from as early as 15 percent into the project."

To determine the earned value, the following steps are performed:

How do I compute earned value and use it to assess progress?

1. The *budgeted cost of work scheduled* (BCWS) is determined for each work task represented in the schedule. During estimation, the work (in person-hours or person-days) of each software engineering task is planned. Hence, $BCWS_i$ is the effort planned for work task *i*. To determine progress at a given point along the project schedule, the value of BCWS is the sum of the $BCWS_i$ values for all work tasks that should have been completed by that point in time on the project schedule.
2. The BCWS values for all work tasks are summed to derive the *budget at completion*, BAC. Hence,

   $BAC = \Sigma\ (BCWS_k)$ for all tasks k
3. Next, the value for *budgeted cost of work performed* (BCWP) is computed. The value for BCWP is the sum of the BCWS values for all work tasks that have actually been completed by a point in time on the project schedule.

Wilkens [WIL99] notes that "the distinction between the BCWS and the BCWP is that the former represents the budget of the activities that were planned to be completed and the latter represents the budget of the activities that actually were completed." Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:

Schedule performance index, SPI = BCWP/BCWS
Schedule variance, SV = BCWP − BCWS

WebRef
A wide array of earned value analysis resources can be found at **www.acq.osd.mil/pm/.**

SPI is an indication of the efficiency with which the project is utilizing scheduled resources. An SPI value close to 1.0 indicates efficient execution of the project schedule. SV is simply an absolute indication of variance from the planned schedule.

Percent scheduled for completion = BCWS/BAC

provides an indication of the percentage of work that should have been completed by time *t*.

Percent complete = BCWP/BAC

provides a quantitative indication of the percent of completeness of the project at a given point in time, *t*.

It is also possible to compute the *actual cost of work performed,* ACWP. The value for ACWP is the sum of the effort actually expended on work tasks that have been completed by a point in time on the project schedule. It is then possible to compute

Cost performance index, CPI = BCWP/ACWP
Cost variance, CV = BCWP − ACWP

A CPI value close to 1.0 provides a strong indication that the project is within its defined budget. CV is an absolute indication of cost savings (against planned costs) or shortfall at a particular stage of a project.

Like over-the-horizon radar, earned value analysis illuminates scheduling difficulties before they might otherwise be apparent. This enables the software project manager to take corrective action before a project crisis develops.

24.7 Summary

Scheduling is the culmination of a planning activity that is a primary component of software project management. When combined with estimation methods and risk analysis, scheduling establishes a road map for the project manager.

Scheduling begins with process decomposition. The characteristics of the project are used to adapt an appropriate task set for the work to be done. A task network depicts each engineering task, its dependency on other tasks, and its projected duration. The task network is used to compute the critical path, a timeline chart and a variety of project information. Using the schedule as a guide, the project manager can track and control each step in the software process.

References

[BRO95] Brooks, M., *The Mythical Man-Month,* anniversary edition, Addison-Wesley, 1995.
[FLE98] Fleming, Q. W., and J. M. Koppelman, "Earned Value Project Management," *Crosstalk,* vol. 11, no. 7, July 1998, p. 19.
[HUM95] Humphrey, W., *A Discipline for Software Engineering,* Addison-Wesley, 1995.
[NOR70] Norden, P., "Useful Tools for Project Management," in *Management of Production,* M. K. Starr, ed., Penguin Books, 1970.

[PAG85] Page-Jones, M., *Practical Project Management*, Dorset House, 1985, pp. 90–91.
[PRE99] Pressman, R. S., *Adaptable Process Model*, R. S. Pressman & Associates, 1999.
[PUT78] Putnam, L., "A General Empirical Solution to the Macro Software Sizing and Estimation Problem," *IEEE Trans. Software Engineering*, vol SE-4, no. 4, July 1978, pp. 345–361.
[PUT92] Putnam, L., and W. Myers, *Measures for Excellence*, Yourdon Press, 1992.
[WIL99] Wilkens, T. T., "Earned Value, Clear and Simple," Primavera Systems, April 1, 1999, p. 2.
[ZAH95] Zahniser, R., "Time-boxing for Top Team Performance," *Software Development*, March 1995, pp. 34–38.

Problems and Points to Ponder

24.1. Assume that you have been contracted by a university to develop an on-line course registration system (OLCRS). First, act as the customer (if you're a student, that should be easy!) and specify the characteristics of a good system. (Alternatively, your instructor will provide you with a set of preliminary requirements for the system.) Using the estimation methods discussed in Chapter 23, develop an effort and duration estimate for OLCRS. Suggest how you would:

a. Define parallel work activities during the OLCRS project.
b. Distribute effort throughout the project.
c. Establish milestones for the project.

24.2. Is there ever a case where a software project milestone is not tied to a review? If so, provide one or more examples.

24.3. Using a scheduling tool (if available) or paper and pencil (if necessary), develop a timeline chart for the OLCRS project.

24.4. The relationship between people and time is highly nonlinear. Using Putnam's software equation (described in Section 24.2.2), develop a table that relates number of people to project duration for a software project requiring 50,000 LOC and 15 person-years of effort (the productivity parameter is 5000). Assume that the software must be delivered in 24 months plus or minus 12 months.

24.5. Although adding people to a late software project can make it later, there are circumstances in which this is not true. Describe them.

24.6. Select an appropriate task set for the OLCRS project.

24.7. "Unreasonable" deadlines are a fact of life in the software business. How should you proceed if you're faced with one?

24.8. "Communication overhead" can occur when multiple people work on a software project. The time spent communicating with others reduces individual productivity (LOC/person-month), and the result is less productivity for the team. Illustrate (quantitatively) how engineers who are well-versed in good software engineering practices and use formal technical reviews can increase the production rate of a team (when compared to the sum of individual production rates). Hint: You can assume that reviews reduce rework and that rework can account for 20–40 percent of a person's time.

24.9. Define a task network for OLCRS, or alternatively, for another software project that interests you. Be sure to show tasks and milestones and to attach effort and duration estimates to each task. If possible, use an automated scheduling tool to perform this work.

24.10. If an automated scheduling tool is available, determine the critical path for the network defined in Problem 24.9.

24.11. What is the difference between a macroscopic schedule and a detailed schedule. Is it possible to manage a project if only a macroscopic schedule is developed? Why?

Full

24.12. Assume you are a software project manager and that you've been asked to compute earned value statistics for a small software project. The project has 56 planned work tasks that are estimated to require 582 person-days to complete. At the time that you've been asked to do the earned value analysis, 12 tasks have been completed. However the project schedule indicates that 15 tasks should have been completed. The following scheduling data (in person-days) are available:

Task	Planned Effort	Actual Effort
1	12.0	12.5
2	15.0	11.0
3	13.0	17.0
4	8.0	9.5
5	9.5	9.0
6	18.0	19.0
7	10.0	10.0
8	4.0	4.5
9	12.0	10.0
10	6.0	6.5
11	5.0	4.0
12	14.0	14.5
13	16.0	—
14	6.0	—
15	8.0	—

Compute the SPI, schedule variance, percent scheduled for completion, percent complete, CPI, and cost variance for the project.

Further Readings and Information Sources

Virtually every book written on software project management contains a discussion of scheduling. The Project Management Institute (*PMBOK Guide*, PMI, 2001), Wysoki and his colleagues (*Effective Project Management*, Wiley, 2000), Lewis (*Project Planning Scheduling and Control*, third edition, McGraw-Hill, 2000), Bennatan (*On Time, Within Budget: Software Project Management Practices and Techniques*, third edition, Wiley, 2000), McConnell (*Software Project Survival Guide*, Microsoft Press, 1998), and Roetzheim and Beasley (*Software Project Cost and Schedule Estimating: Best Practices*, Prentice-Hall, 1997) contain worthwhile discussions of the subject. Boddie (Crunch Mode, Prentice-Hall, 1987) has written a book for all managers who "have 90 days to do a six-month project."

McConnell (*Rapid Development*, Microsoft Press, 1996) presents an excellent discussion of the issues that lead to overly optimistic software project scheduling and what you can do about it. O'Connell (*How to Run Successful Projects II: The Silver Bullet*, Prentice-Hall, 1997) presents a step-by-step approach to project management that will help you develop a realistic schedule for your projects.

Webb and Wake (*Using Earned Value: A Project Manager's Guide*, Ashgate Publishing, 2003) and Fleming and Koppelman (*Earned Value Project Management*, Project Management Institute Publications, 1996) discuss the use of earned value techniques for project planning, tracking, and control in considerable detail.

A wide variety of information sources on software project scheduling is available on the Internet. An up-to-date list of World Wide Web references can be found at the SEPA Web site: **http://www.mhhe.com/pressman.**

CHAPTER

Start //

25 Risk Management

Key Concepts

assessment
identification
projection
refinement
principles
proactive strategy
reactive strategy
risk categories
risk exposure
risk table
RMMM
safety and hazards

In his book on risk analysis and management, Robert Charette [CHA89] presents a conceptual definition of risk:

> First, risk concerns future happenings. Today and yesterday are beyond active concern, as we are already reaping what was previously sowed by our past actions. The question is, can we, therefore, by changing our actions today, create an opportunity for a different and hopefully better situation for ourselves tomorrow. This means, second, that risk involves change, such as in changes of mind, opinion, actions, or places . . . [Third,] risk involves choice, and the uncertainty that choice itself entails. Thus paradoxically, risk, like death and taxes, is one of the few certainties of life.

When risk is considered in the context of software engineering, Charette's three conceptual underpinnings are always in evidence. The future is our concern—what risks might cause the software project to go awry? Change is our concern—how will changes in customer requirements, development technologies, target environments, and all other entities connected to the project affect timeliness and overall success? Last, we must grapple with choices—what methods and tools should we use, how many people should be involved, how much emphasis on quality is "enough"?

Peter Drucker [DRU75] once said, "While it is futile to try to eliminate risk, and questionable to try to minimize it, it is essential that the risks taken be the right risks." Before we can identify the "right risks" to be taken during a software project, it is important to identify all risks that are obvious to both managers and practitioners.

Quick Look

What is it? Risk analysis and management are a series of steps that help a software team to understand and manage uncertainty. Many problems can plague a software project. A risk is a potential problem—it might happen, it might not. But, regardless of the outcome, it's a really good idea to identify it, assess its probability of occurrence, estimate its impact, and establish a contingency plan should the problem actually occur.

Who does it? Everyone involved in the software process—managers, software engineers, and stakeholders—participate in risk analysis and management.

Why is it important? Think about the Boy Scout motto: Be prepared. Software is a difficult undertaking. Lots of things can go wrong, and frankly, many often do. It's for this reason that being prepared—understanding the risks and taking proactive measures to avoid or manage them—is a key element of good software project management.

What are the steps? Recognizing what can go wrong is the first step, called "risk identification." Next, each risk is analyzed to determine the likelihood that it will occur and the damage that it will do if it does occur. Once this information is established, risks are ranked, by probability and impact. Finally, a plan is developed to manage those risks with high probability and high impact.

What is the work product? A risk mitigation, monitoring, and management (RMMM) plan or a set of risk information sheets is produced.

How do I ensure that I've done it right? The risks that are analyzed and managed should be derived from thorough study of the people, the product, the process, and the project. The RMMM plan should be revisited as the project proceeds to ensure that risks are kept up to date. Contingency plans for risk management should be realistic.

25.1 Reactive vs. Proactive Risk Strategies

Reactive risk strategies have been laughingly called the "Indiana Jones school of risk management" [THO92]. In the 1980s-era movies that carried his name, Indiana Jones, when faced with overwhelming difficulty, would invariably say, "Don't worry, I'll think of something!" Never worrying about problems until they happened, Indy would react in some heroic way.

"If you don't actively attack the risks, they will actively attack you."

Tom Gilb

Sadly, the average software project manager is not Indiana Jones, and the members of the software project team are not his trusty sidekicks. Yet, the majority of software teams rely solely on reactive risk strategies. At best, a reactive strategy monitors the project for likely risks. Resources are set aside to deal with them, should they become actual problems. More commonly, the software team does nothing about risks until something goes wrong. Then, the team flies into action in an attempt to correct the problem rapidly. This is often called a *fire-fighting mode.* When this fails, "crisis management" [CHA92] takes over and the project is in real jeopardy.

A considerably more intelligent strategy for risk management is to be proactive. A proactive strategy begins long before technical work is initiated. Potential risks are identified, their probability and impact are assessed, and they are ranked by importance. Then, the software team establishes a plan for managing risk. The primary objective is to avoid risk, but because not all risks can be avoided, the team works to develop a contingency plan that will enable it to respond in a controlled and effective manner. Throughout the remainder of this chapter, we discuss a proactive strategy for risk management.

[illegible]

Although there has been considerable debate about the proper definition for software risk, there is general agreement that risk always involves two characteristics [HIG95]:

- *Uncertainty*—the risk may or may not happen; that is, there are no 100% probable risks.[1]
- *Loss*—if the risk becomes a reality, unwanted consequences or losses will occur.

When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk. To accomplish this, different categories of risks are considered.

What types of risks are we likely to encounter as software is built?

Project risks threaten the project plan. That is, if project risks become real, it is likely that project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project. In Chapter 23, project complexity, size, and the degree of structural uncertainty were also defined as project (and estimation) risk factors.

Technical risks threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible. Technical risks identify potential design, implementation, interface, verification, and maintenance problems. In addition, specification ambiguity, technical uncertainty, technical obsolescence, and "leading-edge" technology are also risk factors. Technical risks occur because the problem is harder to solve than we thought it would be.

Business risks threaten the viability of the software to be built. Business risks often jeopardize the project or the product. Candidates for the top five business risks are (1) building an excellent product or system that no one really wants (market risk), (2) building a product that no longer fits into the overall business strategy for the company (strategic risk), (3) building a product that the sales force doesn't understand how to sell (sales risk), (4) losing the support of senior management due to a change in focus or a change in people (management risk), and (5) losing budgetary or personnel commitment (budget risk).

It is extremely important to note that simple risk categorization won't always work. Some risks are simply unpredictable in advance.

Another general categorization of risks has been proposed by Charette [CHA89]. *Known risks* are those that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date, lack of documented requirements or software scope, poor development environment). *Pre-*

1 A risk that is 100 percent probable is a constraint on the software project.

dictable risks are extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced). *Unpredictable risks* are the joker in the deck. They can and do occur, but they are extremely difficult to identify in advance.

INFO

Seven Principles of Risk Management

The Software Engineering Institute (SEI) (www.sei.cmu.edu) identifies seven principles that "provide a framework to accomplish effective risk management." They are:

Maintain a global perspective—view software risks within the context of system in which it is a component and the business problem that it is intended to solve.

Take a forward-looking view—think about the risks that may arise in the future (e.g., due to changes in the software); establish contingency plans so that future events are manageable.

Encourage open communication—if someone states a potential risk, don't discount it. If a risk is proposed in an informal manner, consider it. Encourage all stakeholders and users to suggest risks at any time.

Integrate—a consideration of risk must be integrated into the software process.

Emphasize a continuous process—the team must be vigilant throughout the software process, modifying identified risks as more information is known and adding new ones as better insight is achieved.

Develop a shared product vision—if all stakeholders share the same vision of the software, it is likely that better risk identification and assessment will occur.

Encourage teamwork—the talents, skills and knowledge of all stakeholders should be pooled when risk management activities are conducted.

25.3 Risk Identification

Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.). By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary.

There are two distinct types of risks for each of the categories that have been presented in Section 25.2: generic risks and product-specific risks. *Generic risks* are a potential threat to every software project. *Product-specific risks* can be identified only by those with a clear understanding of the technology, the people, and the environment that is specific to the software that is to be built. To identify product-specific risks, the project plan and the software statement of scope are examined, and an answer to the following question is developed: "What special characteristics of this product may threaten our project plan?"

"Projects with no real risks are losers. They are almost always devoid of benefit; that's why they weren't done years ago."

Tom DeMarco and Tim Lister

ADVICE

Although generic risks are important to consider, it's the product-specific risks that cause the most headaches. Be certain to spend the time to identify as many product-specific risks as possible.

One method for identifying risks is to create a risk item checklist. The checklist can be used for risk identification and focuses on some subset of known and predictable risks in the following generic subcategories:

- *Product size*—risks associated with the overall size of the software to be built or modified.
- *Business impact*—risks associated with constraints imposed by management or the marketplace.
- *Customer characteristics*—risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.
- *Process definition*—risks associated with the degree to which the software process has been defined and is followed by the development organization.
- *Development environment*—risks associated with the availability and quality of the tools to be used to build the product.
- *Technology to be built*—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
- *Staff size and experience*—risks associated with the overall technical and project experience of the software engineers who will do the work.

The risk item checklist can be organized in different ways. Questions relevant to each of the topics can be answered for each software project. The answers to these questions allow the planner to estimate the impact of risk. A different risk item checklist format simply lists characteristics that are relevant to each generic subcategory. Finally, a set of "risk components and drivers" [AFC88] are listed along with their probability of occurrence. Drivers for performance, support, cost, and schedule are discussed in answer to later questions.

A number of comprehensive checklists for software project risk have been proposed in the literature (e.g., [SEI93], [KAR96]). These provide useful insight into generic risks for software projects and should be used whenever risk analysis and management are instituted. However, a relatively short list of questions [KEI98] can be used to provide a preliminary indication of whether a project is "at risk."

25.3.1 Assessing Overall Project Risk

The following questions have been derived from risk data obtained by surveying experienced software project managers in different parts of the world [KEI98]. The questions are ordered by their relative importance to the success of a project.

Is the software project we're working on at serious risk?

1. Have top software and customer managers formally committed to support the project?
2. Are end-users enthusiastically committed to the project and the system/product to be built?

3. Are requirements fully understood by the software engineering team and its customers?
4. Have customers been involved fully in the definition of requirements?
5. Do end-users have realistic expectations?
6. Is project scope stable?
7. Does the software engineering team have the right mix of skills?
8. Are project requirements stable?
9. Does the project team have experience with the technology to be implemented?
10. Is the number of people on the project team adequate to do the job?
11. Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

WebRef
Risk radar is a database and tools that help managers identify, rank, and communicate project risks. It can be found at **www.spmn.com.**

"Risk management is project management for adults."
Tim Lister

If any one of these questions is answered negatively, mitigation, monitoring, and management steps should be instituted without fail. The degree to which the project is at risk is directly proportional to the number of negative responses to these questions.

25.3.2 Risk Components and Drivers

The U.S. Air Force [AFC88] has written a pamphlet that contains excellent guidelines for software risk identification and abatement. The Air Force approach requires that the project manager identify the risk drivers that affect software risk components—performance, cost, support, and schedule. In the context of this discussion, the risk components are defined in the following manner:

- *Performance risk*—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.
- *Cost risk*—the degree of uncertainty that the project budget will be maintained.
- *Support risk*—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.
- *Schedule risk*—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

The impact of each risk driver on the risk component is divided into one of four impact categories—negligible, marginal, critical, or catastrophic. Referring to Figure 25.1 [BOE89], a characterization of the potential consequences of errors (rows labeled 1) or a failure to achieve a desired outcome (rows labeled 2) are described. The impact category is chosen based on the characterization that best fits the description in the table.

FIGURE 25.1

Impact assessment [BOE89]

Category \ Components		Performance	Support	Cost	Schedule
Catastrophic	1	Failure to meet the requirement would result in mission failure		Failure results in increased costs and schedule delays with expected values in excess of $500K	
	2	Significant degradation to nonachievement of technical performance	Nonresponsive or unsupportable software	Significant financial shortages, budget overrun likely	Unachievable IOC
Critical	1	Failure to meet the requirement would degrade system performance to a point where mission success is questionable		Failure results in operational delays and/or increased costs with expected value of $100K to $500K	
	2	Some reduction in technical performance	Minor delays in software modifications	Some shortage of financial resources, possible overruns	Possible slippage in IOC
Marginal	1	Failure to meet the requirement would result in degradation of secondary mission		Costs, impacts, and/or recoverable schedule slips with expected value of $1K to $100K	
	2	Minimal to small reduction in technical performance	Responsive software support	Sufficient financial resources	Realistic, achievable schedule
Negligible	1	Failure to meet the requirement would create inconvenience or nonoperational impact		Error results in minor cost and/or schedule impact with expected value of less than $1K	
	2	No reduction in technical performance	Easily supportable software	Possible budget underrun	Early achievable IOC

Note: (1) The potential consequence of undetected software errors or faults.
(2) The potential consequence if the desired outcome is not achieved.

25.4 Risk Projection

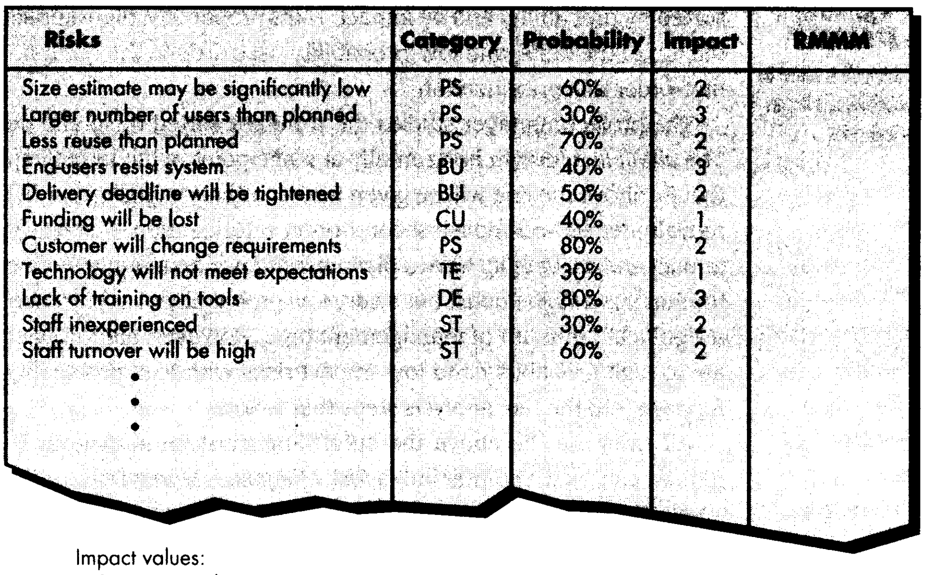
Risk projection, also called *risk estimation,* attempts to rate each risk in two ways—(1) the likelihood or probability that the risk is real and (2) the consequences of the problems associated with the risk, should it occur. The project planner, along with other managers and technical staff, performs four risk projection steps:

1. Establish a scale that reflects the perceived likelihood of a risk.
2. Delineate the consequences of the risk.
3. Estimate the impact of the risk on the project and the product.
4. Note the overall accuracy of the risk projection so that there will be no misunderstandings.

The intent of these steps is to consider risks in a manner that leads to prioritization. No software team has the resources to address every possible risk with the same de-

FIGURE 25.2

Sample risk table prior to sorting



Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
•				
•				
•				

Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

gree of rigor. By prioritizing risks, the team can allocate resources where they will have the most impact.

25.4.1 Developing a Risk Table

A risk table provides a project manager with a simple technique for risk projection.[2] A sample risk table is illustrated in Figure 25.2.

ADVICE

Think hard about the software you're about to build and ask, yourself, what can go wrong? Create your own list and ask other members of the team to do the same.

A project team begins by listing all risks (no matter how remote) in the first column of the table. This can be accomplished with the help of the risk item checklists referenced in Section 25.3. Each risk is categorized in the second column (e.g., PS implies a project size risk, BU implies a business risk). The probability of occurrence of each risk is entered in the next column of the table. The probability value for each risk can be estimated by team members individually. Individual team members are polled in round-robin fashion until their assessment of risk probability begins to converge.

Next, the impact of each risk is assessed. Each risk component is assessed using the characterization presented in Figure 25.1, and an impact category is determined. The categories for each of the four risk components—performance, support, cost, and schedule—are averaged[3] to determine an overall impact value.

2 The risk table can be implemented as a spreadsheet model. This enables easy manipulation and sorting of the entries.

3 A weighted average can be used if one risk component has more significance for a project.

KEY POINT
A risk table is sorted by probability and impact to rank risks.

Once the first four columns of the risk table have been completed, the table is sorted by probability and by impact. High-probability, high-impact risks percolate to the top of the table, and low-probability risks drop to the bottom. This accomplishes first-order risk prioritization.

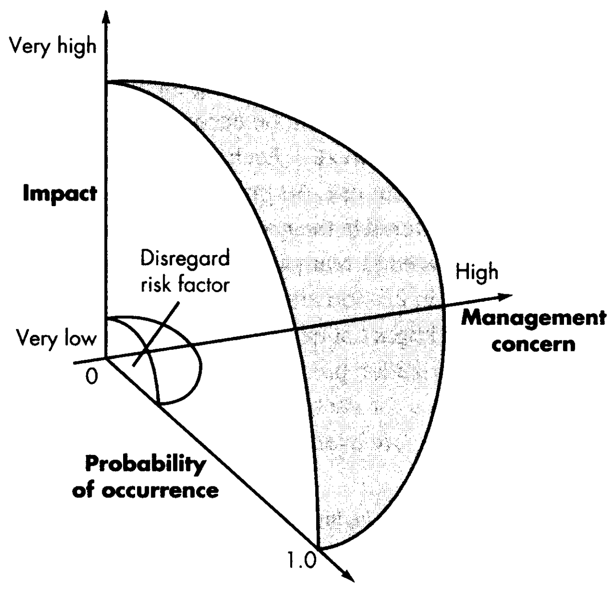
The project manager studies the resultant sorted table and defines a cutoff line. The *cutoff line* (drawn horizontally at some point in the table) implies that only risks that lie above the line will be given further attention. Risks that fall below the line are reevaluated to accomplish second-order prioritization. Referring to Figure 25.3, risk impact and probability have a distinct influence on management concern. A risk factor that has a high impact but a very low probability of occurrence should not absorb a significant amount of management time. However, high-impact risks with moderate to high probability and low-impact risks with high probability should be carried forward into the risk analysis steps that follow.

All risks that lie above the cutoff line must be managed. The column labeled RMMM contains a pointer into a *Risk Mitigation, Monitoring, and Management Plan* or alternatively, a collection of *risk information sheets* developed for all risks that lie above the cutoff. The RMMM plan and risk information sheets are discussed in Sections 25.5 and 25.6.

"[Today,] no one has the luxury of getting to know a task so well that it holds no surprises, and surprises mean risk."
Stephen Grey

Risk probability can be determined by making individual estimates and then developing a single consensus value. Although that approach is workable, more so-

FIGURE 25.3
Risk and management concern



phisticated techniques for determining risk probability have been developed [AFC88]. Risk drivers can be assessed on a qualitative probability scale that has the following values: impossible, improbable, probable, and frequent. Mathematical probability can then be associated with each qualitative value (e.g., a probability of 0.7 to 0.95 implies a highly probable risk).

25.4.2 Assessing Risk Impact

Three factors affect the consequences that are likely if a risk does occur: its nature, scope, and timing. The nature of the risk indicates the problems that are likely if it occurs. For example, a poorly defined external interface to customer hardware (a technical risk) will preclude early design and testing and will likely lead to system integration problems late in a project. The scope of a risk combines the severity (just how serious is it?) with its overall distribution (how much of the project will be affected, or how many customers are harmed?). Finally, the timing of a risk considers when and for how long the impact will be felt. In most cases, a project manager might want the "bad news" to occur as soon as possible, but in some cases, the longer the delay, the better.

Returning once more to the risk analysis approach proposed by the U.S. Air Force [AFC88], the following steps are recommended to determine the overall consequences of a risk:

How do we assess the consequences of a risk?

1. Determine the average probability of occurrence value for each risk component.
2. Using Figure 25.1, determine the impact for each component based on the criteria shown.
3. Complete the risk table and analyze the results as described in the preceding sections.

The overall *risk exposure,* RE, is determined using the following relationship [HAL98]:

$$\text{RE} = P \times C$$

where P is the probability of occurrence for a risk, and C is the cost to the project should the risk occur.

For example, assume that the software team defines a project risk in the following manner:

Risk identification. Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

Risk probability. 80 percent (likely).

Risk impact. 60 reusable software components were planned. If only 70 percent can be used, 18 components would have to be developed from scratch (in addition to other custom software that has been scheduled for development). Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is \$14.00, the overall cost (impact) to develop the components would be $18 \times 100 \times 14 = \$25,200$.

Risk exposure. $RE = 0.80 \times 25,200 \sim \$20,200$.

ADVICE

Compare RE for all risks to the cost estimate for the project. If RE is greater than 50 percent of the project cost, the viability of the project must be reevaluated.

Risk exposure can be computed for each risk in the risk table, once an estimate of the cost of the risk is made. The total risk exposure for all risks (above the cutoff line in the risk table) can provide a means for adjusting the final cost estimate for a project. It can also be used to predict the probable increase in staff resources required at various points during the project schedule.

The risk projection and analysis techniques described in Sections 25.4.1 and 25.4.2 are applied iteratively as the software project proceeds. The project team should revisit the risk table at regular intervals, reevaluating each risk to determine when new circumstances cause its probability and impact to change. As a consequence of this activity, it may be necessary to add new risks to the table, remove some risks that are no longer relevant, and change the relative positions of others.

SAFEHOME

Risk Analysis

The scene: Doug Miller's office, prior to the initiation of the *SafeHome* software project.

The players: Doug Miller (manager of the *SafeHome* software engineering team) and Vinod Raman, Jamie Lazar, and other members of the product software engineering team.

The conversation:

Doug: I'd like to spend some time brainstorming risks for the *SafeHome* project.

Jamie: As in what can go wrong?

Doug: Yep. Here are a few categories where things can go wrong. [He shows everyone the categories noted in the introduction to Section 25.3.]

Vinod: Umm . . . do you want us to just call them out, or . . .

Doug: No here's what I thought we'd do. Everyone make a list of risks . . . right now . . .

(Ten minutes pass; everyone is writing.)

Doug: Okay, stop.

Jamie: But I'm not done!

Doug: That's okay. We'll revisit the list again. Now, for each item on your list, assign a percent likelihood that the risk will occur. Then, assign an impact to the project on a scale of 1 (minor) to 5 (catastrophic).

Vinod: So if I think that the risk is a coin flip, I specify a 50 percent likelihood, and if I think it'll have a moderate project impact, I specify a 3, right?

Doug: Exactly.

(Five minutes pass; everyone is writing.)

Doug: Okay, stop. Now we'll make a group list on the white board. I'll do the writing, we'll call out one entry from your list in round robin format.

(Fifteen minutes pass; the list is created.)

Jamie (pointing at the board and laughing): Vinod, that risk (pointing toward an entry on the board) is ridiculous. There's a higher likelihood that we'll all get hit by lightning. We should remove it.

Doug: No, let's leave it for now. We consider all risks, no matter how weird. Later we'll winnow the list.

Jamie: But we already have over 40 risks . . . how on earth can we manage them all?

Doug: We can't. That's why we'll define a cut-off after we sort these guys. I'll do that off-line, and we'll meet again tomorrow. For now, get back to work . . . and in your spare time, think about any risks that we've missed.

25.5 Risk Refinement

During early stages of project planning, a risk may be stated quite generally. As time passes and more is learned about the project and the risk, it may be possible to refine the risk into a set of more detailed risks, each somewhat easier to mitigate, monitor, and manage.

What's a good way to describe a risk?

One way to do this is to represent the risk in *condition-transition-consequence* (CTC) format [GLU94]. That is, the risk is stated in the following form:

Given that <condition> then there is concern that (possibly) <consequence>.

Using the CTC format for the reuse risk noted in Section 25.4.2, we can write:

Given that all reusable software components must conform to specific design standards and that some do not conform, then there is concern that (possibly) only 70 percent of the planned reusable modules may actually be integrated into the as-built system, resulting in the need to custom engineer the remaining 30 percent of components.

This general condition can be refined in the following manner:

Subcondition 1. Certain reusable components were developed by a third party with no knowledge of internal design standards.

Subcondition 2. The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.

Subcondition 3. Certain reusable components have been implemented in a language that is not supported on the target environment.

The consequences associated with these refined subconditions remain the same (i.e., 30 percent of software components must be custom engineered), but the refinement helps to isolate the underlying risks and might lead to easier analysis and response.

25.6 Risk Mitigation, Monitoring, and Management

All of the risk analysis activities presented to this point have a single goal—to assist the project team in developing a strategy for dealing with risk. An effective strategy must consider three issues:

- Risk avoidance.
- Risk monitoring.
- Risk management and contingency planning.

If a software team adopts a proactive approach to risk, avoidance is always the best strategy. This is achieved by developing a plan for risk mitigation. For example, assume that high staff turnover is noted as a project risk, r_1. Based on past history and management intuition, the likelihood, l_1, of high turnover is estimated to be 0.70 (70 percent, rather high) and the impact, x_1, is projected as critical. That is, high turnover will have a critical impact on project cost and schedule.

"If I take so many precautions, it is because I leave nothing to chance."
Napoleon

To mitigate this risk, project management must develop a strategy for reducing turnover. Among the possible steps to be taken are:

What can we do to mitigate a risk?

- Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market).
- Mitigate those causes that are under our control before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
- Organize project teams so that information about each development activity is widely dispersed.
- Define documentation standards and establish mechanisms to ensure that documents are developed in a timely manner.
- Conduct peer reviews of all work (so that more than one person is "up to speed").
- Assign a backup staff member for every critical technologist.

As the project proceeds, risk monitoring activities commence. The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely. In the case of high staff turnover, the following factors can be monitored:

- General attitude of team members based on project pressures.
- The degree to which the team has jelled.
- Interpersonal relationships among team members.
- Potential problems with compensation and benefits.
- The availability of jobs within the company and outside it.

In addition to monitoring these factors, a project manager should monitor the effectiveness of risk mitigation steps. For example, a risk mitigation step noted earlier called for the definition of documentation standards and mechanisms to be sure that documents are developed in a timely manner. This is one mechanism for ensuring continuity, should a critical individual leave the project. The project manager should monitor documents carefully to ensure that each can stand on its own and that each imparts information that would be necessary if a newcomer were forced to join the software team somewhere in the middle of the project.

Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality. Continuing the example, the project is well underway, and a number of people announce that they will be leaving. If the mitigation strategy has been followed, backup is available, information is documented, and knowledge has been dispersed across the team. In addition, the project manager may temporarily refocus resources (and readjust the project schedule) to those functions that are fully staffed, enabling newcomers who must be added to the team to "get up to speed." Those individuals who are leaving are asked to stop all work and spend their last weeks in "knowledge transfer mode." This might include video-based knowledge capture, the development of "commentary documents," and/or meeting with other team members who will remain on the project.

ADVICE

If RE for a specific risk is less than the cost of risk mitigation, don't try to mitigate the risk but continue to monitor it.

It is important to note that risk mitigation, monitoring, and management (RMMM) steps incur additional project cost. For example, spending the time to "backup" every critical technologist costs money. Part of risk management, therefore, is to evaluate when the benefits accrued by the RMMM steps are outweighed by the costs associated with implementing them. In essence, the project planner performs a classic cost/benefit analysis. If risk aversion steps for high turnover will increase both project cost and duration by an estimated 15 percent, but the predominant cost factor is "backup," management may decide not to implement this step. On the other hand, if the risk aversion steps are projected to increase costs by 5 percent and duration by only 3 percent, management will likely put all into place.

For a large project, 30 or 40 risks may be identified. If between three and seven risk management steps are identified for each, risk management may become a project in itself! For this reason, we adapt the Pareto 80-20 rule to software risk. Experience indicates that 80 percent of the overall project risk (i.e., 80 percent of the potential for project failure) can be accounted for by only 20 percent of the identified risks. The work performed during earlier risk analysis steps will help the planner to determine which of the risks reside in that 20 percent (e.g., risks that lead to the highest risk exposure). For this reason, some of the risks identified, assessed, and projected may not make it into the RMMM plan—they don't fall into the critical 20 percent (the risks with highest project priority).

WebRef

A voluminous archive containing all entries from the ACM Forum on Risks to the Public can be found at **catless.ncl.ac.uk/Risks.**

Risk is not limited to the software project itself. Risks can occur after the software has been successfully developed and delivered to the customer. These risks are typically associated with the consequences of software failure in the field.

Software safety and hazard analysis [LEV95] are software quality assurance activities (Chapter 26) that focus on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail. If hazards can be identified early in the software engineering process, software design features can be specified that will either eliminate or control potential hazards.

25.7 The RMMM Plan

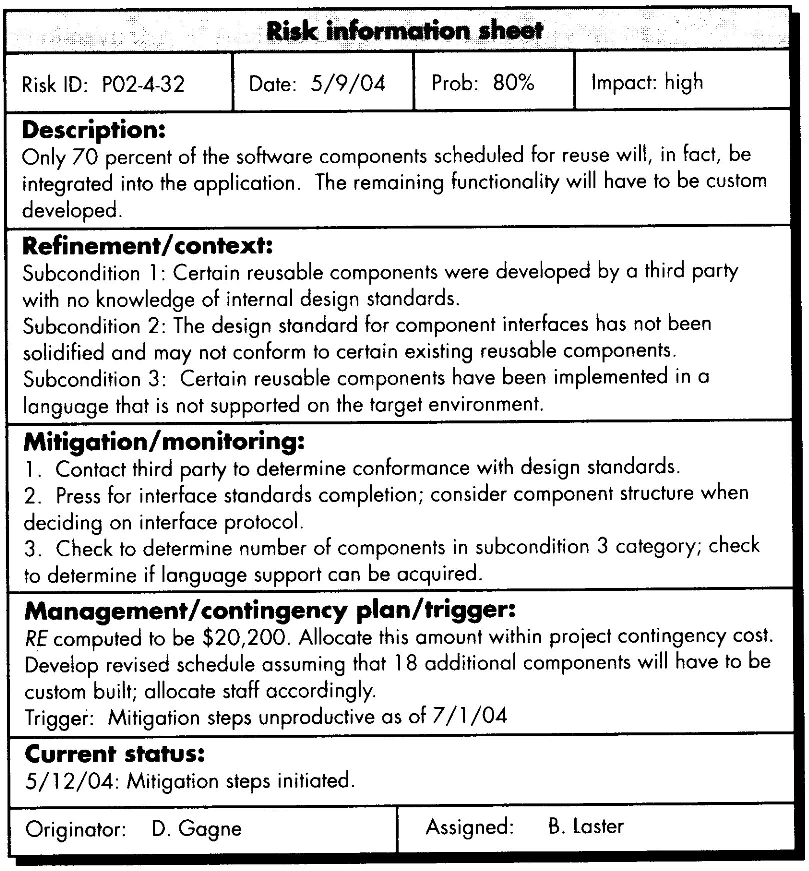
A risk management strategy can be included in the software project plan or the risk management steps can be organized into a separate *Risk Mitigation, Monitoring and Management Plan.* The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan.

Some software teams do not develop a formal RMMM document. Rather, each risk is documented individually using a *risk information sheet* (RIS) [WIL97]. In most cases, the RIS is maintained using a database system, so that creation and information entry, priority ordering, searches, and other analysis may be accomplished easily. The format of the RIS is illustrated in Figure 25.4.

Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence. As we have already discussed, risk mitigation is a problem avoidance activity. Risk monitoring is a project tracking activity with three

FIGURE 25.4

Risk information sheet [WIL97]



Risk information sheet

Risk ID: P02-4-32	Date: 5/9/04	Prob: 80%	Impact: high

Description:
Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

Refinement/context:
Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards.
Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.
Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.

Mitigation/monitoring:
1. Contact third party to determine conformance with design standards.
2. Press for interface standards completion; consider component structure when deciding on interface protocol.
3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.

Management/contingency plan/trigger:
RE computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly.
Trigger: Mitigation steps unproductive as of 7/1/04

Current status:
5/12/04: Mitigation steps initiated.

Originator: D. Gagne	Assigned: B. Laster

primary objectives: (1) to assess whether predicted risks do, in fact, occur; (2) to ensure that risk aversion steps defined for the risk are being properly applied; and (3) to collect information that can be used for future risk analysis. In many cases, the problems that occur during a project can be traced to more than one risk. Another job of risk monitoring is to attempt to allocate origin (what risk(s) caused which problems throughout the project).

SOFTWARE TOOLS

Risk Management

Objective: The objective of risk management tools is to assist a project team in defining risks, assessing their impact and probability, and tracking risks throughout a software project.

Mechanics: In general, risk management tools assist in generic risk identification by providing a list of typical project and business risks, providing checklists or other "interview" techniques that assist in identifying project specific risks, assigning probability and impact to each risk, supporting risk mitigation strategies, and generating many different risk-related reports.

Representative Tools[4]

Riskman, developed at Arizona State University (www.eas.asu.edu/~sdm/merrill/riskman.html), is a risk evaluation expert system that identifies project-related risks.

Risk Radar, developed by SPMN (www.spmn.com), assists project managers in identifying and managing project risks.

RiskTrak, developed by RST (www.risktrac.com), supports the identification, analysis, reporting, and management of risks throughout a software project.

Risk+, developed by C/S Solutions (www.CS-solutions.com), integrates with Microsoft Project to quantify cost and schedule uncertainty.

X:PRIMER, developed by GrafP Technologies (www.grafp.com), is a generic Web-based tool that predicts what can go wrong on a project and identifies root causes for potential failures and effective countermeasures.

25.8 SUMMARY

Whenever a lot is riding on a software project, common sense dictates risk analysis. And yet, most software project managers do it informally and superficially, if they do it at all. The time spent identifying, analyzing, and managing risk pays itself back in many ways: less upheaval during the project, a greater ability to track and control a project, and the confidence that comes with planning for problems before they occur.

Risk analysis can absorb a significant amount of project planning effort. Identification, projection, assessment, management, and monitoring all take time. But the effort is worth it. To quote Sun Tzu, a Chinese general who lived 2500 years ago, "If you know the enemy and know yourself, you need not fear the result of a hundred battles." For the software project manager, the enemy is risk.

4 Tools noted here do not represent an endorsement, but rather a sampling of tools in this category. In most cases, tool names are trademarked by their respective developers.

References

[AFC88] *Software Risk Abatement,* AFCS/AFLC Pamphlet 800-45, U.S. Air Force, September 30, 1988.

[BOE89] Boehm, B. W., *Software Risk Management,* IEEE Computer Society Press, 1989.

[CHA89] Charette, R. N., *Software Engineering Risk Analysis and Management,* McGraw-Hill/Intertext, 1989.

[CHA92] Charette, R. N., "Building Bridges over Intelligent Rivers," *American Programmer,* vol. 5, no. 7, September, 1992, pp. 2–9.

[DRU75] Drucker, P., *Management,* W. H. Heinemann, 1975.

[GIL88] Gilb, T., *Principles of Software Engineering Management,* Addison-Wesley, 1988.

[GLU94] Gluch, D. P., "A Construct for Describing Software Development Risks," CMU/SEI-94-TR-14, Software Engineering Institute, 1994.

[HAL98] Hall, E. M., *Managing Risk: Methods for Software Systems Development,* Addison-Wesley, 1998.

[HIG95] Higuera, R. P., "Team Risk Management," *CrossTalk,* U.S. Dept. of Defense, January 1995, pp. 2–4.

[KAR96] Karolak, D. W., *Software Engineering Risk Management,* IEEE Computer Society Press, 1996.

[KEI98] Keil, M., et al., "A Framework for Identifying Software Project Risks," *CACM,* vol. 41, no. 1, November 1998, pp. 76–83.

[LEV95] Leveson, N. G., *Safeware: System Safety and Computers,* Addison-Wesley, 1995.

[SEI93] "Taxonomy-Based Risk Identification," Software Engineering Institute, CMU/SEI-93-TR-6, 1993.

[THO92] Thomsett, R., "The Indiana Jones School of Risk Management," *American Programmer,* vol. 5, no. 7, September 1992, pp. 10–18.

[WIL97] Williams, R. C., J. A. Walker, and A. J. Dorofee, "Putting Risk Management into Practice," *IEEE Software,* May 1997, pp. 75–81.

Problems and Points to Ponder

25.1. Develop a risk monitoring strategy and specific risk monitoring activities for three of the risks noted in Figure 25.2. Be sure to identify the factors that you'll be monitoring to determine whether the risk is becoming more or less likely.

25.2. You've been asked to build software to support a low-cost video editing system. The system accepts digital video as input, stores the video on disk, and then allows the user to do a wide range of edits to the digitized video. The result can then be output to DVD or other media. Do a small amount of research on systems of this type, and then make a list of technology risks that you would face as you begin a project of this type.

25.3. Add three additional questions or topics to each of the risk item checklists presented at the SEPA Web site.

25.4. Develop a risk mitigation strategy and specific risk mitigation activities for three of the risks noted in Figure 25.2.

25.5. Provide five examples from other fields that illustrate the problems associated with a reactive risk strategy.

25.6. Describe the difference between "known risks" and "predictable risks."

25.7. Describe the difference between risk components and risk drivers.

25.8. You're the project manager for a major software company. You've been asked to lead a team that's developing "next generation" word-processing software. Create a risk table for the project.

25.9. Attempt to refine three of the risks noted in Figure 25.2 and then create risk information sheets for each.

25.10. Can you think of a situation in which a high-probability, high-impact risk would not be considered as part of your RMMM plan?

25.11. Represent three of the risks noted in Figure 25.2 using a CTC format.

25.12. Recompute the risk exposure discussed in Section 25.4.2 when cost/LOC is $16 and the probability is 60 percent.

25.13. Develop a risk management strategy and specific risk management activities for three of the risks noted in Figure 25.2.

25.14. Describe five software application areas in which software safety and hazard analysis would be a major concern.

Further Readings and Information Sources

The software risk management literature has expanded significantly over the past decade. DeMarco and Lister (*Dancing with Bears,* Dorset House, 2003) have written an entertaining and insightful book that guides software managers and practitioners through risk management. Moynihan (*Coping with IT/IS Risk Management,* Springer-Verlag, 2002) presents pragmatic advice from project managers who deal with risk on a continuing basis. Royer (*Project Risk Management,* Management Concepts, 2002) and Smith and Merritt (*Proactive Risk Management,* Productivity Press, 2002) suggest a proactive process for risk management. Karolak(*Software Engineering Risk Management,* Wiley, 2002) has written a guidebook that introduces an easy-to-use risk analysis model with worthwhile checklists and questionnaires supported by a software package.

Schuyler (*Risk and Decision Analysis in Projects,* PMI, 2001) considers risk analysis from a statistical perspective. Hall (*Managing Risk: Methods for Software Systems Development,* Addison-Wesley, 1998) presents one of the more thorough treatments of the subject. Myerson (*Risk Management Processing for Software Engineering Models,* Artech House, 1997) considers metrics, security, process models and other topics. A useful snapshot of risk assessment has been written by Grey (*Practical Risk Assessment for Project Management,* Wiley, 1995). His abbreviated treatment provides a good introduction to the subject.

Capers Jones (*Assessment and Control of Software Risks,* Prentice-Hall, 1994) presents a detailed discussion of software risks that includes data collected from hundreds of software projects. Jones defines 60 risk factors that can affect the outcome of software projects. Boehm [BOE89] suggests excellent questionnaire and checklist formats that can prove invaluable in identifying risk. Charette [CHA89] presents a detailed treatment of the mechanics of risk analysis, calling on probability theory and statistical techniques to analyze risks. In a companion volume, Charette (*Application Strategies for Risk Analysis,* McGraw-Hill, 1990) discusses risk in the context of both system and software engineering and suggests pragmatic strategies for risk management. Gilb (*Principles of Software Engineering Management,* Addison-Wesley, 1988) presents a set of "principles" (which are often amusing and sometimes profound) that can serve as a worthwhile guide for risk management.

Ewusi-Mensah (*Software Development Failures: Anatomy of Abandoned Projects,* MIT Press, 2003) and Yourdon (*Death March,* Prentice-Hall, 1997) discuss what happens when risks overwhelm a software project team. Bernstein (*Against the Gods,* Wiley, 1998) presents an entertaining history of risk that goes back to ancient times.

The Software Engineering Institute has published many detailed reports and guidebooks on risk analysis and management. The Air Force Systems Command pamphlet AFSCP 800-45 [AFC88] describes risk identification and reduction techniques. Every issue of the *ACM Software Engineering Notes* has a section entitled "Risks to the Public" (editor, P. G. Neumann). If you want the latest and best software horror stories, this is the place to go.

A wide variety of information sources on software risk management is available on the Internet. An up-to-date list of World Wide Web references can be found at the SEPA Web site: **http://www.mhhe.com/pressman.**

CHAPTER

26 QUALITY MANAGEMENT

KEY CONCEPTS
cost of quality
defect amplification
defect costs
ISO 9001: 2000
reliability
reviews (FTRs)
quality
quality control
sampling
six sigma
software safety
SQA Plan
statistical SQA

The software engineering approach described in this book works toward a single goal: to produce high-quality software. Yet many readers will be challenged by the question: What is software quality?

Philip Crosby [CRO79], in his landmark book on quality, provides a wry answer to this question:

> The problem of quality management is not what people don't know about it. The problem is what they think they do know . . .
>
> In this regard, quality has much in common with sex. Everybody is for it. (Under certain conditions, of course.) Everyone feels they understand it. (Even though they wouldn't want to explain it.) Everyone thinks execution is only a matter of following natural inclinations. (After all, we do get along somehow.) And, of course, most people feel that problems in these areas are caused by other people. (If only they would take the time to do things right.)

Some software developers continue to believe that software quality is something you begin to worry about after code has been generated. Nothing could be further from the truth! *Quality management* (often called *software quality assurance*) is an umbrella activity (Chapter 2) that is applied throughout the software process.

Quality management encompasses (1) a software quality assurance (SQA) process; (2) specific quality assurance and quality control tasks (including formal technical reviews and a multitiered testing strategy); (3) effective software engineering practice (methods and tools); (4) control of all software work products

QUICK LOOK

What is it? It's not enough to talk the talk by saying that software quality is important. You have to (1) explicitly define what is meant when you say "software quality," (2) create a set of activities that will help ensure that every software engineering work product exhibits high quality, (3) perform quality control and assurance activities on every software project, (4) use metrics to develop strategies for improving your software process and, as a consequence, the quality of the end product.

Who does it? Everyone involved in the software engineering process is responsible for quality.

Why is it important? You can do it right, or you can do it over again. If a software team stresses quality in all software engineering activities, it reduces the amount of rework that it must do. That results in lower costs, and more importantly, improved time-to-market.

What are the steps? Before software quality assurance activities can be initiated, it is important to define "software quality" at a number of different levels of abstraction. Once you understand what quality is, a software team must identify a set of SQA activities that will filter errors out of work products before they are passed on.

What is the work product? A *Software Quality Assurance Plan* is created to define a software team's SQA strategy. During analysis, design, and code generation, the primary SQA work product is the formal technical review summary report. During testing, test plans and procedures are produced. Other work products associated with process improvement may also be generated.

How do I ensure that I've done it right? Find errors before they become defects! That is, work to improve your defect removal efficiency (Chapter 22), thereby reducing the amount of rework that your software team has to perform.

and the changes made to them (Chapter 27); (5) a procedure to ensure compliance with software development standards (when applicable), and (6) measurement and reporting mechanisms.

In this chapter, we focus on the management issues and the process-specific activities that enable a software organization to ensure that it does the right things at the right time in the right way.

26.1 Quality Concepts[1]

KEY POINT
Controlling variation is the key to a high-quality product. In the software context, we strive to control the variation in the generic process we apply and the quality emphasis that permeates software engineering work.

Variation control is the heart of quality control. A manufacturer wants to minimize the variation among the products that are produced, even when doing something relatively simple like duplicating DVDs. Surely, this cannot be a problem—duplicating DVDs is a trivial manufacturing operation, and we can guarantee that exact duplicates of the software are always created.

Or can we? We need'to ensure the tracks are placed on the DVDs within a specified tolerance so that the overwhelming majority of DVD drives can read the media. The disk duplication machines can, and do, wear and go out of tolerance. So even a "simple" process such as DVD duplication may encounter problems due to variation between samples.

But how does this apply to software work? How might a software development organization need to control variation? From one project to another, we want to minimize the difference between the predicted resources needed to complete a project and the actual resources used, including staff, equipment, and calendar time. In general, we would like to make sure our testing program covers a known percentage of the software, from one release to another. Not only do we want to minimize the number of defects that are released to the field, we'd like to ensure that the variance in the number of bugs is also minimized from one release to another. (Our customers will likely be upset if the third release of a product has 10 times as many

1 This section, written by Michael Stovsky, has been adapted from "Fundamentals of ISO 9000," a workbook developed for *Essential Software Engineering*, a video curriculum developed by R. S. Pressman & Associates, Inc. Reprinted with permission.

defects as the previous release.) We would like to minimize the differences in speed and accuracy of our hotline support responses to customer problems. The list goes on and on.

26.1.1 Quality

The *American Heritage Dictionary* defines *quality* as "a characteristic or attribute of something." As an attribute of an item, quality refers to measurable characteristics—things we can compare to known standards such as length, color, electrical properties, and malleability. However, software, largely an intellectual entity, is more challenging to characterize than physical objects.

Nevertheless, measures of a program's characteristics do exist. These properties include cyclomatic complexity, cohesion, number of function points, lines of code, and many others discussed in Chapter 15. When we examine an item based on its measurable characteristics, two kinds of quality may be encountered: quality of design and quality of conformance.

Quality of design refers to the characteristics that designers specify for an item. *Quality of conformance* is the degree to which the design specifications are followed during manufacturing.

"People forget how fast you did a job—but they always remember how well you did it."
Howard Newton

In software development, quality of design encompasses requirements, specifications, and the design of the system. Quality of conformance is an issue focused primarily on implementation. If the implementation follows the design and the resulting system meets its requirements and performance goals, conformance quality is high.

But are quality of design and quality of conformance the only issues that software engineers must consider? Robert Glass [GLA98] argues that a more "intuitive" relationship is in order:

$$\begin{aligned}\text{user satisfaction} = {} & \text{compliant product} + \text{good quality} \\ & + \text{delivery within budget and schedule}\end{aligned}$$

At the bottom line, Glass contends that quality is important, but if the user isn't satisfied, nothing else really matters. DeMarco [DEM99] reinforces this view when he states: "A product's quality is a function of how much it changes the world for the better." This view of quality contends that if a software product provides substantial benefit to its end-users, they may be willing to tolerate occasional reliability or performance problems.

26.1.2 Quality Control

? What is software quality control?

Variation control may be equated to quality control. But how do we achieve quality control? Quality control involves the series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it. Quality control includes a feedback loop to the process that created the work product. The combination of measurement and feedback allows us to tune the process when the work products created fail to meet their specifications.

A key concept of quality control is that all work products have defined, measurable specifications to which we may compare the output of each process. The feedback loop is essential to minimize the defects produced.

26.1.3 Quality Assurance

WebRef
Useful links to SQA resources can be found at **www.qualitytree.com/links/index.htm.**

Quality assurance consists of a set of auditing and reporting functions that assess the effectiveness and completeness of quality control activities. The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals. Of course, if the data provided through quality assurance identify problems, it is management's responsibility to address the problems and apply the necessary resources to resolve quality issues.

26.1.4 Cost of Quality

The cost of quality includes all costs incurred in the pursuit of quality or in performing quality-related activities. Cost of quality studies are conducted to provide a baseline for the current cost of quality, identify opportunities for reducing the cost of quality, and provide a normalized basis of comparison. The basis of normalization is almost always dollars. Once we have normalized quality costs on a dollar basis, we have the necessary data to evaluate where the opportunities lie to improve our processes. Furthermore, we can evaluate the effect of changes in dollar-based terms.

What are the components of the cost of quality?

Quality costs may be divided into costs associated with prevention, appraisal, and failure. *Prevention costs* include quality planning, formal technical reviews, test equipment, and training. *Appraisal costs* include activities to gain insight into product condition the "first time through" each process. Examples of appraisal costs include in-process and interprocess inspection, equipment calibration and maintenance, and testing.

ADVICE
Don't be afraid to incur significant prevention costs. Rest assured that your investment will provide an excellent return.

Failure costs are those that would disappear if no defects appeared before shipping a product to customers. Failure costs may be subdivided into internal failure costs and external failure costs. *Internal failure costs* are incurred when we detect a defect in our product prior to shipment. Internal failure costs include rework, repair, and failure mode analysis. *External failure costs* are associated with defects found after the product has been shipped to the customer. Examples of external failure costs are complaint resolution, product return and replacement, help line support, and warranty work.

As expected, the relative costs to find and repair a defect increase dramatically as we go from prevention to detection to internal failure to external failure costs.

FIGURE 26.1

Relative cost of correcting an error

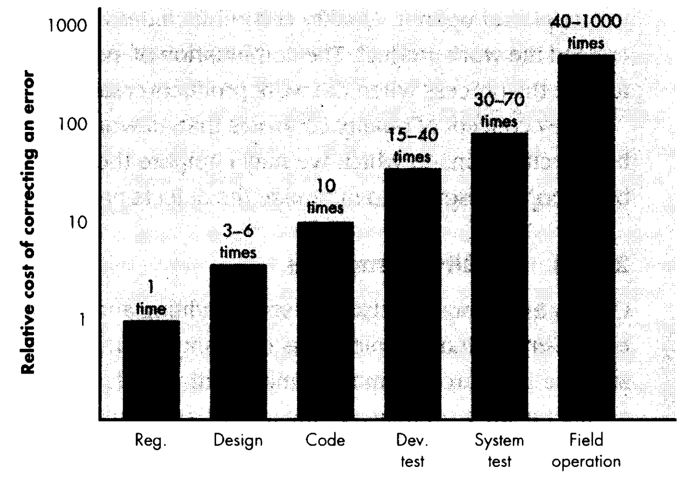


Figure 26.1, based on data collected by Boehm [BOE81] and others, illustrates this phenomenon.

"It takes less time to do a thing right than to explain why you did it wrong."

H. W. Longfellow

26.2 SOFTWARE QUALITY ASSURANCE

Even the most jaded software developers will agree that high-quality software is an important goal. But how do we define quality? A wag once said, "Every program does something right, it just may not be the thing that we want it to do."

Many definitions of software quality have been proposed in the literature. For our purposes, *software quality* is defined as:

? How do we define software quality?

Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

There is little question that this definition could be modified or extended. In fact, the definition of software quality could be debated endlessly. For the purposes of this book, this definition serves to emphasize three important points:

1. Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.
2. Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.

3. A set of implicit requirements often goes unmentioned (e.g., the desire for ease of use and good maintainability). If software conforms to its explicit requirements but fails to meet implicit requirements, software quality is suspect.

26.2.1 Background Issues

Quality control and assurance are essential activities for any business that produces products to be used by others. Prior to the twentieth century, quality control was the sole responsibility of the craftsperson who built a product. The first formal quality assurance and control function was introduced at Bell Labs in 1916 and spread rapidly throughout the manufacturing world. During the 1940s, more formal approaches to quality control were suggested. These relied on measurement and continuous process improvement [DEM86] as key elements of quality management.

"You made too many wrong mistakes."

Yogi Berra

Today, every company has mechanisms to ensure quality in its products. In fact, explicit statements of a company's concern for quality have become a marketing ploy during the past few decades.

The history of quality assurance in software development parallels the history of quality in hardware manufacturing. During the early days of computing (1950s and 1960s), quality was the sole responsibility of the programmer. Standards for quality assurance for software were introduced in military contract software development during the 1970s and have spread rapidly into software development in the commercial world [IEE94]. Extending the definition presented earlier, software quality assurance is a "planned and systematic pattern of actions" [SCH98] that are required to ensure high quality in software. Many different constituencies have software quality assurance responsibility—software engineers, project managers, customers, salespeople, and the individuals who serve within an SQA group.

The SQA group serves as the customer's in-house representative. That is, the people who perform SQA must look at the software from the customer's point of view. Does the software adequately meet the quality factors noted in Chapter 15? Has software development been conducted according to preestablished standards? Have technical disciplines properly performed their roles as part of the SQA activity? The SQA group attempts to answer these and other questions to ensure that software quality is maintained.

26.2.2 SQA Activities

Software quality assurance is composed of a variety of tasks associated with two different constituencies—the software engineers who do technical work and an SQA group that has responsibility for quality assurance planning, oversight, record keeping, analysis, and reporting.

Software engineers address quality (and perform quality assurance and quality control activities) by applying solid technical methods and measures, conducting formal technical reviews, and performing well-planned software testing. Only reviews are discussed in this chapter. Technology topics are discussed in Parts 1, 2, 3, and 5 of this book.

The charter of the SQA group is to assist the software team in achieving a high-quality end product. The Software Engineering Institute recommends a set of SQA activities that address quality assurance planning, oversight, record keeping, analysis, and reporting. These activities are performed (or facilitated) by an independent SQA group that conducts the following activities:

What is the role of an SQA group?

Prepares an SQA plan for a project. The plan is developed during project planning and is reviewed by all stakeholders. Quality assurance activities performed by the software engineering team and the SQA group are governed by the plan. The plan identifies evaluations to be performed, audits and reviews to be performed, standards that are applicable to the project, procedures for error reporting and tracking, documents to be produced by the SQA group, and amount of feedback provided to the software project team.

Participates in the development of the project's software process description. The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g., ISO-9001), and other parts of the software project plan.

Reviews software engineering activities to verify compliance with the defined software process. The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made.

Audits designated software work products to verify compliance with those defined as part of the software process. The SQA group reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made; and periodically reports the results of its work to the project manager.

Ensures that deviations in software work and work products are documented and handled according to a documented procedure. Deviations may be encountered in the project plan, process description, applicable standards, or technical work products.

Records any noncompliance and reports to senior management. Noncompliance items are tracked until they are resolved.

In addition to these activities, the SQA group coordinates the control and management of change (Chapter 27) and helps to collect and analyze software metrics.