SPECIAL FEATURE:

Impact of Schedule Estimation on Software Project Behavior

Tarek K. Abdel-Hamid, SRI International
Stuart E. Madnick, Massachusetts Institute of Technology

Efforts to develop better estimation tools must address two issues: First, are more accurate tools necessarily better? Second, how can we measure a new estimation method's accuracy?

Schedule estimation has historically been, and continues to be, a major difficulty in managing software development projects.[1] Farquhar articulated this problem's significance:

> Unable to estimate accurately, the manager can know with certainty neither what resources to commit to an effort nor, in retrospect, how well these resources were used. The lack of a firm foundation for these two judgments can reduce programming management to a random process in that positive control is next to impossible. This situation often results in the budget overruns and schedule slippages that are all too common.[2]

Over the last decade, a number of quantitative software estimation models have been proposed. They range from theoretical models to empirical ones. An empirical model uses data from previous projects to evaluate the current project and derives basic formulas from analyses of available historical databases. In contrast, a theoretical model uses formulas based on global assumptions, such as the rate at which people solve problems and the number of problems awaiting solution at a given point.

Still, software cost and schedule estimation continue to be major difficulties. As noted by Mohanty:

> Even today, almost no model can estimate the true cost of software with any degree of accuracy.[3]

Research efforts attempting to develop "better" estimation tools need to address two important issues: first, whether a more accurate estimation tool is necessarily a better tool; second, how can we adequately measure the accuracy of a new estimation method. Our objective in this article is to address these two issues.

An ongoing research effort to study the dynamics of software development resulted in our system dynamics model of software development project management. This model served as a laboratory vehicle for simulating the impact of schedule estimation on software project behavior. Our study revealed two interesting facts: (1) different estimates create different projects, implying that new software estimation tools cannot be adequately judged by how accurately they estimate historical projects; and (2) more accurate estimates are not necessarily better estimates.

Different estimates create different projects

In this section we are concerned about the impact of alternative schedule estimations rather than the methodology used to arrive at estimates. In later sections we will give actual examples of methodologies used. For now, the reader is merely asked to assume that two different methods exist (a simplistic method A could be "man-days needed = number of pages of specifications × 10 man-days per page" and a simplistic method B could be "man-days needed = number of words in specifications × 0.1 man-days per word").

Consider the following scenario: A 64,000-delivered-source-instructions (DSI) software project that had been estimated at its initiation to take 2359 man-days (using estimation method A) ends up actually consuming 3795 man-days. The project's specifications are then fed into estimation method B (that is being considered by management to replace method A) and its results compared to the project's

0740-7459/86/0700/0070$01.00 © 1986 IEEE

actual performance. Let us assume that method B produces a 5900-man-day estimate. If we define "percent of relative absolute error" in estimating man-days (MD) as

$$\%\ \text{Error} = 100 * \text{ABS}[MD_{Actual} - MD_{Estimate}] / MD_{Actual}$$

then, for estimation method A,

$$\%\ \text{Error}_A = 100 * \text{ABS}[3795 - 2359] / 3795 = 38\%$$

and for method B,

$$\%\ \text{Error}_B = 100 * \text{ABS}[3795 - 5900] / 3795 = 55\%$$

Can one conclude from this that method B would have provided a less accurate estimate of the project's man-days had it been used instead of method A? The answer is No; we cannot draw such a conclusion. Had the project been initiated with B's 5900-man-day estimate instead of A's 2359-man-day estimate, we cannot assume it would have still consumed 3795 actual man-days. In fact, the project could end up consuming much more or much less than 3795 man-days. And before such a determination can be made, no accurate assessment of A's versus B's relative accuracy can be made.

Different estimates create different projects. Koolhass states the principle as follows:

> When experimenting with the system about which we are trying to obtain knowledge, we create a new system.[4]

Koolhass goes on to illustrate the principle with an anecdote: A man inquires through the bedroom door of his sick friend: "How are you?"—whereupon his friend replies "Fine!" and the effort kills him.

This phenomenon is somewhat analogous to Heisenberg's uncertainty principle in experimentation. By imposing different estimates on a software project we would, in a real sense, be creating different projects. The next section explains how.

The feedback impact of project estimates

Research findings indicate that decisions people make in project situations, and actions they choose to take, are significantly influenced by pressures and perceptions the project schedule produces.[5] Figure 1's causal loop diagram, excerpted from Abdel-Hamid,[6] depicts such schedule influences.

Schedules have a direct influence on hiring and firing decisions throughout a software project's life. In TRW's COCOMO model,[7] for example, the project's average staff size would be determined by dividing the man-day estimate by the development time estimate (TDEV). Thus, a tight time schedule (that is, a low TDEV value) means a larger work force. Also, scheduling can dramatically change manpower loading throughout the life of a project. For example, the work force level in some environments shoots upwards towards the end of a late project when there are strict constraints on the extent to which the project's schedule is allowed to slip.

Through its effects on the work force level, a project's schedule also affects productivity (as illustrated in Figure 1). A higher work force level means more communication and training overhead, affecting productivity negatively.

Productivity is also influenced by the presence of any man-day shortages. If a project is considered behind schedule (that is, when the total effort needed to complete the project is seen as greater than the total effort actually remaining in the project's budget), software developers tend to work harder; they allocate more man-hours to the project in an attempt to compensate for the perceived shortage and bring the project back on schedule.

Obviously, such man-day shortages are more prone to occur when projects are initially underestimated. Conversely, man-day "excesses" could arise if project management initially overestimates a project; as a result, the project would be perceived ahead of schedule (that is, the total man-days remaining in the project's budget would exceed what the project members perceive is needed to complete the project). When such a situation occurs,

> Parkinson's law indicates that people will use the extra time for . . . personal activities, catching up on the mail, etc.[7]

Of course, this means that they become less productive.

Having identified how software project estimation can influence project behavior, are we now in a position to return to the scenario presented earlier and answer the still unanswered question—namely, whether estimation method A is truly more accurate than method B?

Identifying feedback relationships through which project estimation influences project behavior is one thing; discerning dynamic implications of such interactions on the total system is another. Paraphrasing Richardson and Pugh:[8] The behavior of interconnected feedback loop systems often confounds intuition and analysis, even though the dynamic implications of isolated loops may be reasonably obvious.

One option is to effect a controlled experiment conducting the 64,000-DSI software project twice under exactly the

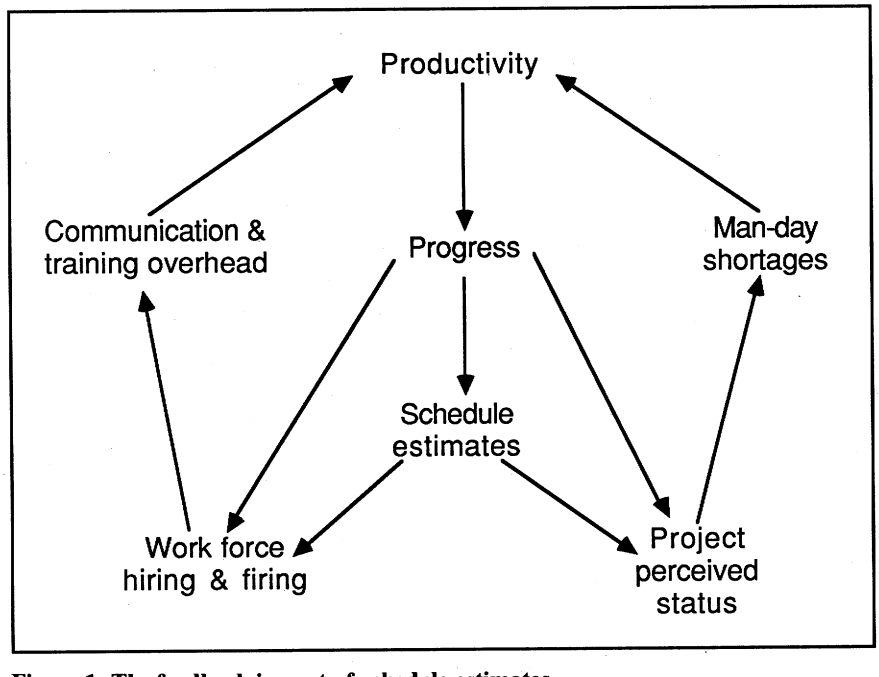

Figure 1. The feedback impact of schedule estimates.

same conditions—except that in one case it would be initiated with a 2359-man-day estimate (the method A basis), and in the second case a 5900-man-day estimate would be used (the method B basis). While theoretically possible, such an option is usually infeasible from a practical point of view because of its high cost in terms of both money and time.

Simulation experimentation provides a more attractive alternative. In addition to permitting less costly and less time-consuming experimentation, simulation makes perfectly controlled experiments possible. Of course, such simulations can only capture elements of the model and not reality, but this type of simulation has been found helpful in understanding the behavior of complex social systems.[8]

A software management system dynamics computer model

Research findings reported in this article are based on a doctoral research effort at MIT's Sloan School of Management studying software development dynamics[6] culminating in the development of a system dynamics simulation model of software development project management. This structural model achieved several research objectives, one of which was to serve as a laboratory vehicle for conducting experimentation in software estimation. A major advantage of such a structural model is that it can be used to predict the effect of changes to the development process.[1] This section provides an overview of the model.

Figure 2 depicts the model's four subsystems, namely,

(1) the human resource management subsystem;
(2) the software production subsystem;
(3) the controlling subsystem; and
(4) the planning subsystem.

Figure 2 also illustrates some interrelationships of the four subsystems.

The human resource management subsystem captures hiring, training, assimilation, and transfer of the project's human resource. Such actions are not carried out in a vacuum; as Figure 2 suggests, they affect and are affected by the other subsystems. For example, the project's hiring rate is a function of the work force needed to complete the project on its planned completion date. Similarly, the work force available has direct bearing on the allocation of manpower among the different software production activities in the software production subsystem.

The four primary software production activities are development, quality assurance, rework, and testing. Development comprises both design and coding of the software. As the software is developed, it is also reviewed; for example, structured walk-throughs are used to detect any design/coding errors. Errors detected through such quality assurance activities are then reworked. Not all errors are detected and reworked, however. Some escape detection until the end of development (that is, until the testing phase).

Progress is reported as it is made. A comparison of the project's actual status (versus where it should be according to plan) is a control-type activity captured within the controlling subsystem. Once project status is assessed using available information, it becomes an important input to the planning function.

In the planning subsystem, initial project estimates are made to start the project. Those estimates are revised, when necessary, throughout the project's life. For example, to handle a project that is considered behind schedule, plans can be revised to hire more people, extend the schedule, or do a little of both.

The preceding overview highlights the structure of the model used. Other reports[6,9] provide a full description of the model and of the validation experiments performed on it.

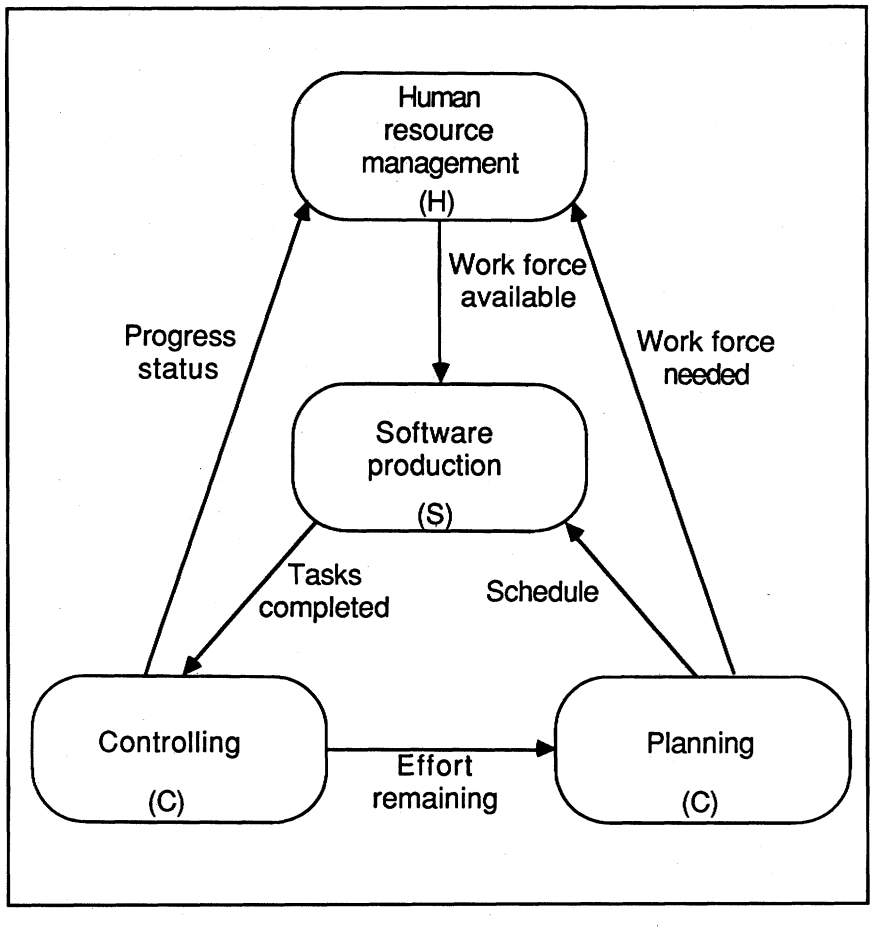

Figure 2. Overview of a software development system dynamics model.

Simulation experiment

The scenario for our simulation experiment is a real-life software development environment—that of a major minicomputer manufacturer involved in our research effort. In this organization, project managers were rewarded based upon how closely their project estimates matched actual project results. The estimation procedure that they informally used follows:

(1) Use basic COCOMO to estimate the number of man-days; that is, use

$$MD = 2.4 * 19 * (KDSI)^{1.05} \text{ man-days}$$

where KDSI is the perceived project size in thousands of delivered source instructions.

(2) Multiply this estimate by a safety fac-

tor (the safety factor ranged from 25 to 50 percent) and add it to MD; that is,

$MD' = (1 + \text{safety factor}) * MD.$

(3) Use the new value of man-days (MD ′) to calculate the development time (TDEV), using COCOMO; that is, use

$TDEV = 47.5 * (MD'/19)^{0.38}$ days.

Notice that the primary input to COCOMO is the perceived (not the real) size of the project in KDSI—since at the beginning of development when estimates are made, the real size of the project is often not known.[7] For our purposes, it is also important to note that COCOMO is used only to exemplify a schedule estimation tool; the structural model developed is not tied to any particular schedule estimation technique.

The safety factor philosophy is in no way unique to our scenario organization. For example, in his study of software cost estimation at the Air Force Systems Command Electronics Systems Division, Devenny[10] found that most program managers budget additional funds for software as a "management reserve." He also found that these management reserves ranged in size (as a percentage of the estimated software cost) from 5 to 50 percent, with a mean of 18 percent. And as was the case with the organization we studied, the policy was informal:

> . . . frequently the reserve was created by the program office with funds not placed on any particular contract. Most of the respondents indicated that the reserve was not identified as such to prevent its loss during a budget cut.[10]

To test the efficacy of various safety factor policies, we ran a number of simulations on a prototypical software project that we will call project Example. Project Example's actual size is 64,000 DSI. At its initiation, however, it was incorrectly considered 42.88 KDSI in size (that is, 33 percent smaller than 64 KDSI). This incorrectly perceived project size of 42.88 KDSI was the input used in COCOMO's estimation equations. The basic COCOMO estimate for man-days (with no safety factor) was

$MD = 2.4 * 19 * (42.88)^{1.05} = 2359$ man-days.

This estimate corresponds to the method A estimate presumed at the beginning of this article.

We experimented with safety factor values ranging from 0 (the base run) to 100 percent. For a 50-percent safety factor, the following estimates would be used:

(1) Calculate MD ′ from MD

$MD' = MD * (1+\text{safety factor}/100)$
$= MD * 1.5 = 3538.5$ man-days.

(2) Calculate TDEV

$TDEV = 47.5 * (MD'/19)^{0.38} = 346$ days.

Figures 3 through 6 exhibit the results of these simulations.

In Figure 3, the "percent relative error" in estimating man-days is plotted against different values of the safety factor. Notice that the safety factor policy seems to be working—the larger the safety factor, the smaller the estimation error. In the 25- to 50-percent range in particular (the same as was used in the scenario organization), the estimation error drops from approximately 40 percent in the base run to values in the upper 20's. In fact, Figure 3 suggests that project managers might not be going far enough by using a 25- to 50-percent safety factor since a 100-percent safety factor

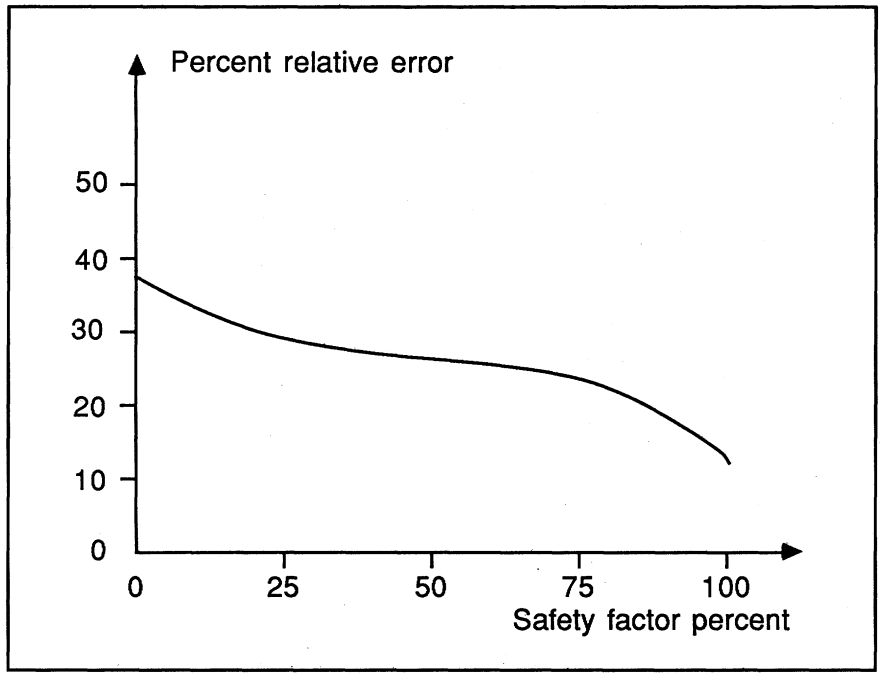

Figure 3. The percentage of relative error in estimating actual man-days.

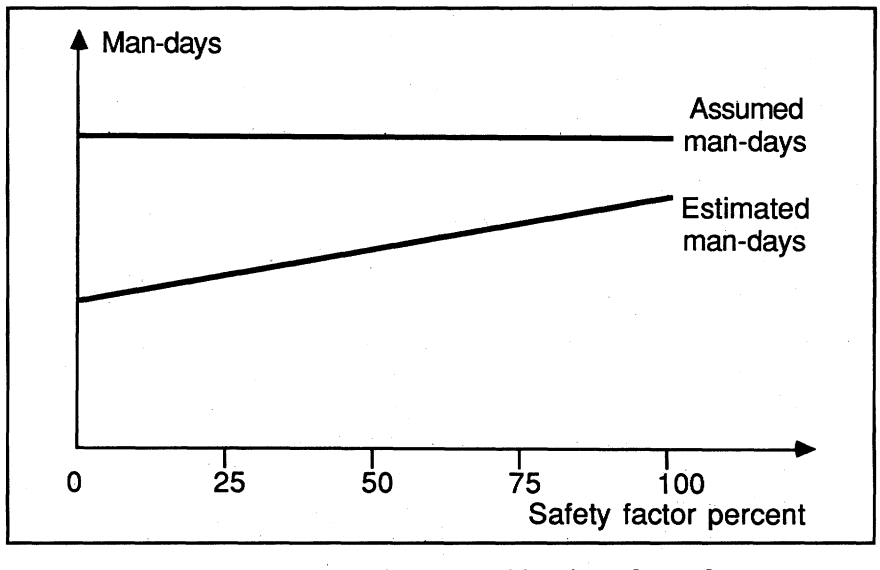

Figure 4. A comparison of assumed man-days with estimated man-days.

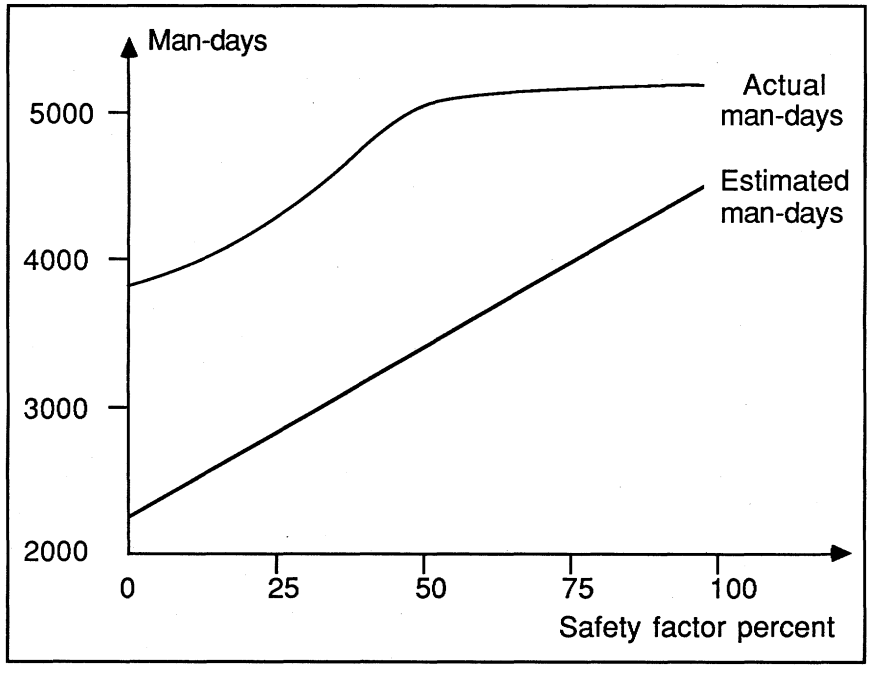

Figure 5. A comparison of actual man-days with estimated man-days.

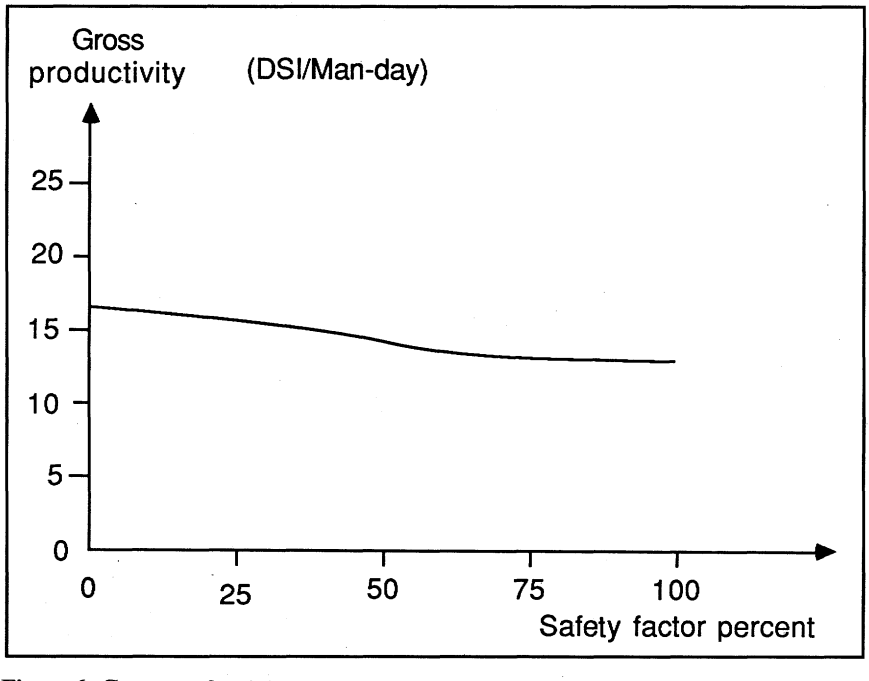

Figure 6. Gross productivity.

would drop the estimation error down to a more rewarding 12 percent.

The rationale for using a safety factor is based on the following assumptions:

(1) Past experience indicates a strong bias among software developers to underestimate the scope of software projects.[10,11]

(2) One might think biases are the easiest of estimating problems to correct since they involve errors moving always in the same direction. But as DeMarco suggests,[11] biases are almost by definition invisible; the same psychological mechanism that creates the bias (for example, the optimism of software developers) works to conceal it.

(3) To rectify such bias, project management uses a safety factor. Pietrasanta[12] observes that when project managers add contingency factors (ranging, say, from 25 to 100 percent), they are saying in essence: I don't know all that is going to happen, so I'll estimate what I don't know as a percentage of what I do know.

In other words, the assumption is that safety factors are simply mechanisms to bring initial man-day estimates closer to true project size in man-days (see Figure 4). Such an assumption cannot be contested solely on the basis of Figure 3, which provides only part of the story. Figure 5 presents a more complete picture; here, we used the model to calculate the actual man-days consumed by the project Example when different safety factors were applied to its initial estimate. The Figure 4 assumption is obviously invalidated. As we use higher safety factors, leading to increasingly generous initial man-day allocations, the actual amount of man-days consumed does not remain at some inherently defined value. In the base run, for example, project Example would be initiated with a man-day estimate of 2359 man-days and would consume 3795 man-days. When a 50-percent safety factor is used, leading to a 3538-man-day initial estimate, Example ends up consuming not 3795 man-days but 5080 man-days.

To reiterate a point made earlier: Different estimates create different projects. Initial project estimates create pressures and perceptions affecting how people behave on the project. In particular, overestimates of project man-days can lead to a larger work force buildup and higher communication and training overhead, ultimately reducing productivity. Furthermore, overestimating a project often invites the expansion of discretionary activity (such as nonproject communication, or personal pursuits) leading to further productivity reductions.

Figure 6 illustrates a plot of gross productivity, defined as the project size in DSI (that is, 64,000 DSI) divided by the actual number of man-days expended for the different safety factor situations. Gross productivity drops from a value of 16.8 DSI/man-day in the base run to as low as 12 DSI/man-day when we use a 100-percent safety factor. The drop in productivity is initially significant, and then levels off for higher safety factors. The reason for this is that, when the safety factor increases from 0 (in the base run) to a relatively small value (perhaps 25 percent), most resulting man-day excesses will be absorbed by employees in the form of less overworking (that is, less days that employees work longer-than-usual hours) and/or more discretionary time.

In the base case, using no safety factor,

backlogs are experienced as project Example consumes more man-days (3795) than budgeted (2359). Using a small safety factor, though, project Example's backlogs will decrease—leading to less overwork durations. As the safety factor is increased further, man-day excesses rather than backlog reductions will result. When these excesses are reasonable, they tend to be absorbed in the form of reasonably expanded discretionary activities; consequently, the project team becomes less productive.

However, there are limits on how much "fat" employees would be willing (or allowed) to absorb. Beyond these limits, man-day excesses would be translated not into less productivity but into cuts in the project's work force, its schedule, or both.[6] As safety factors increase to larger and larger values, productivity losses due to expanded slack time activity decrease, leading to lower drops in gross productivity.

Methods A and B recompared

We can now answer the question posed at the beginning of this article. Our opening scenario concerned a 64,000-DSI project—our own project Example, in fact—and a comparison of two estimation methods. Method A, the estimate used in the base run, produced a 2359-man-day estimate. Since project Example actually consumed 3795 man-days, method A's relative absolute error in estimating man-days is 38 percent. We wondered whether method B, producing a 5900-man-day estimate for project Example (55 percent higher than base-run Example's 3795-man-day expenditure), would have provided a less accurate project estimate had it been used instead of method A.

Method B's estimate of 5900 man-days is 150 percent higher than A's 2359 estimate; indeed, method B is equivalent to a 150-percent safety factor. To check the behavior of project Example had it been estimated using method B, we reran the model with a 5900-man-day estimate.

Using this estimate, project Example consumed 5412 man-days—for a nine-percent error factor. Since method A had a 38-percent error factor, method B proves to be a more accurate estimator. However, method B's improved accuracy is costly—the project consumes 43 percent more man-days!

In terms of the minicomputer manufacturer we studied, the message is clear. While the safety factor policy does achieve its intended objective (namely, producing more accurate estimates), the organization is paying dearly for this accuracy. In terms of man-days, as Figure 5 indicates, a 25- to 50-percent safety factor results in a 15- to 35-percent increase in project cost. Taking an extreme case, method B's 150-percent safety factor would be appropriate for a development manager required to complete a project within 10 percent of estimate. Unfortunately, this would not necessarily be economical for the company since costs would be significantly higher than with method A.

The primary purpose of our research effort was to gain a better understanding of software project management dynamics. We gained two basic insights from the work described in this article:

(1) Different estimates create different projects—implying that project managers as well as students of software estimation should reject the notion that a software estimation tool can be adequately judged based on how accurately it estimates historical projects.

(2) More accurate estimates are not necessarily better estimates—an estimation method should not be judged only on how accurate it is; it should also be judged on how costly are the projects it creates. □

Acknowledgments

We appreciate the contribution of each and every individual in the organizations providing perspectives and data to this research effort. We thank the reviewers, whose suggestions have improved this article's readability. Work described herein was supported, in part, by NASA research grant NAGW-448.

References

1. M.R. Barbacci, A.N. Habermann, and M. Shaw, "The Software Engineering Institute: Bridging Practice and Potential," *IEEE Software,* Vol. 2, No. 6, Nov., 1985, pp. 4-21.
2. J.A. Farquhar, "A Preliminary Inquiry into the Software Estimation Process," Tech. Report, AD F12 052, Defense Documentation Center, Alexandria, Va., Aug. 1970.
3. S.N. Mohanty, "Software Cost Estimation: Present and Future," *Software-Practice and Experience,* Vol. 11, 1981, pp. 103-121.
4. Z. Koolhass, *Organization Dissonance and Change,* John Wiley & Sons, New York, N.Y., 1982.
5. F.P. Brooks, *The Mythical Man-Month,* Addison-Wesley, Reading, Mass., 1978.
6. T.K. Abdel-Hamid, "The Dynamics of Software Development Project Management: An Integrative System Dynamics Perspective," PhD dissertation NTIS N84 16824, MIT, Cambridge, Mass., 1984.
7. B.W. Boehm, *Software Engineering Economics,* Prentice-Hall, Englewood Cliffs, N.J., 1981.
8. G.P. Richardson and G.L. Pugh III, *Introduction to System Dynamics Modeling with Dynamo,* The MIT Press, Cambridge, Mass., 1981.
9. T.K. Abdel-Hamid and S.E. Madnick, *Software Project Management,* Prentice-Hall, Englewood Cliffs, N.J., (scheduled to be published 1986).
10. T.J. Devenny, "An Exploration Study of Software Cost Estimating at the Electronics Systems Division," NTIS, U.S. Department of Commerce, Washington, DC, July 1976.
11. T. DeMarco, *Controlling Software Projects,* Yourdon Press, New York, N.Y., 1982.
12. A.M. Pietrasanta, "Managing the Economics of Computer Programming, *Proc. ACM Nat'l Conf.,* 1968, pp. 341-346.

Tarek K. Abdel-Hamid has been a senior management systems consultant at SRI International since January, 1984. His primary research interests are in software engineering project management, particularly computer-based tools for managing large software projects. He received his BSc in aeronautical engineering from Cairo University in 1972, and his PhD in management information systems from MIT in 1984. Abdel-Hamid is a member of the ACM, the IEEE-CS, and the SIM.

His address is 19 Trillium La., San Carlos, CA 94070.

Stuart Madnick is an associate professor of management at the Massachusetts Institute of Technology. His research interests focus on advanced information systems, database computers, computer architecture, operating systems, and software project management. He is author or coauthor of four books and more than 100 papers and technical reports on these subjects.

Madnick has served as chairman of the IEEE Technical Committee on Database Engineering and as a member of the IEEE Computer Society Board of Governors. He is currently associate editor of the *ACM Transactions on Database Systems.* He received his BS in electrical engineering, his MS in management, and his PhD in computer science from MIT.

His address is MIT Sloan School of Management, E53-317, 50 Memorial Dr., Cambridge, MA 02139.