IMPACT OF SCHEDULE ESTIMATION ON SOFTWARE PROJECT BEHAVIOR

Tarek K. Abdel-Hamid

Stuart E. Madnick

June 1985
(Revised November 1985)

CISR WP No. 127
Sloan WP No. 1685-85

Center for Information Systems Research
Massachusetts Institute of Technology
Sloan School of Management
77 Massachusetts Avenue
Cambridge, Massachusetts, 02139

IMPACT OF SCHEDULE ESTIMATION ON SOFTWARE PROJECT BEHAVIOR

Tarek K. Abdel-Hamid

Stuart E. Madnick

June 1985
(Revised November 1985)

CISR WP No. 127
Sloan WP No.1685-85

This paper has been accepted for Publication in *IEEE Software*.

Center for Information Systems Research
Sloan School of Management
Massachusetts Institute of Technology

IMPACT OF SCHEDULE ESTIMATION ON SOFTWARE PROJECT BEHAVIOR

Abstract

Schedule estimation of software development projects has historically been, and continues to be, a major difficulty associated with the management of software development. 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, and secondly, how to adequately measure the accuracy of a (new) estimation method.

Our objective in this paper is to address these two issues. A series of industry interviews and literature study culminated in the development of a system dynamics model of software development project management. The model served as a laboratory vehicle for conducting simulation experiments on the impact of schedule estimation on software project behavior. The study produced two interesting results: first, that a different estimate "creates" a different project. The important implication that follows from this is that (new) software estimation tools cannot be adequately judged on the basis of how accurately they can estimate historical projects. Secondly, that a more accurate estimate is not necessarily a "better" estimate.

Keywords: Software Project Management, Schedule Estimation, Programmer Behavior, Simulation Model, System Dynamics.

Introduction:

Schedule estimation of software development projects has historically been, and continues to be, a major difficulty associated with the management of software development (Barbacci et al, 1985). Farquhar (1970) articulated the significance of the problem:

> 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 judgements 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...

Over the last decade, a number of quantitative software estimation models have been proposed. They range from theoretical ones, to empirical ones, such as Boehm's COCOMO model (Boehm, 1981). An empirical model uses data from previous projects to evaluate the current project and derives the basic formulae from analysis of the historical data base available. A theoretical model, on the other hand, uses formulae based upon global assumptions, such as the rate at which people solve problems, the number of problems available for solutions at a given point in time, etc.

Still, software cost and schedule estimation continues to be a major difficulty associated with the management of software development. As noted by Mohanty (1981): "Even today, almost no model can estimate the true cost of software with any degree of accuracy".

Most ongoing research efforts are premised on two "intuitive" assumptions: First, that a more accurate estimation tool is a "better" tool, and secondly, that the accuracy of a (new) estimation model can be adequately measured on the basis of how close the model is in matching historical project results. Our objective in this paper is to question these two "intuitive" premises.

Different Estimations Create Different Projects:

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

Consider the following scenario: A 64,000 Delivered Source Instructions (DSI) software project which had been estimated at its initiation, using estimation method "A," to be 2,359 man-days, ends up actually consuming, at its completion, 3,795 man-days. The project's specifications (e.g., its size, complexity, etc.) are then fed into another estimation method "B" (e.g., that is being considered by management for future adoption in place of method "A") and its results compared to the project's actual performance. And let us assume that method "B" produces a 5,900 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,"

$$\% \text{ Error}_A = 100 * \text{ABS}[3,795 - 2,359] / 3,795$$
$$= 38\%$$

And for method "B,"

$$\% \text{ Error}_B = 100 * \text{ABS}[3,795 - 5,900] / 3,795$$
$$= 55\%$$

Question: Can one conclude from this that estimation 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 5,900 man-day estimate, instead of A's 2,359 man-day estimate, we cannot assume it would have still ended up actually consuming exactly 3,795 man-days. In fact the project could end-up consuming much more or much less than 3,795 man-days. And before such a determination can be made, no "accurate" assessment of the relative accuracy of the two methods can be made.

The point we are trying to make is this: a different estimate creates a different project.

The principle can be stated as follows: "When experimenting with the system about which we are trying to obtain knowledge, we create a new system" (Koolhass, 1982). Koolhass gives a fine example of this: "A man who inquires through the door of the bedroom where his friend is sick, "How are you?" whereupon his friend replied "fine," and the effort kills him." This phenomenon is somewhat analogous to the Heisenberg Uncertainty Principle in experimentation.

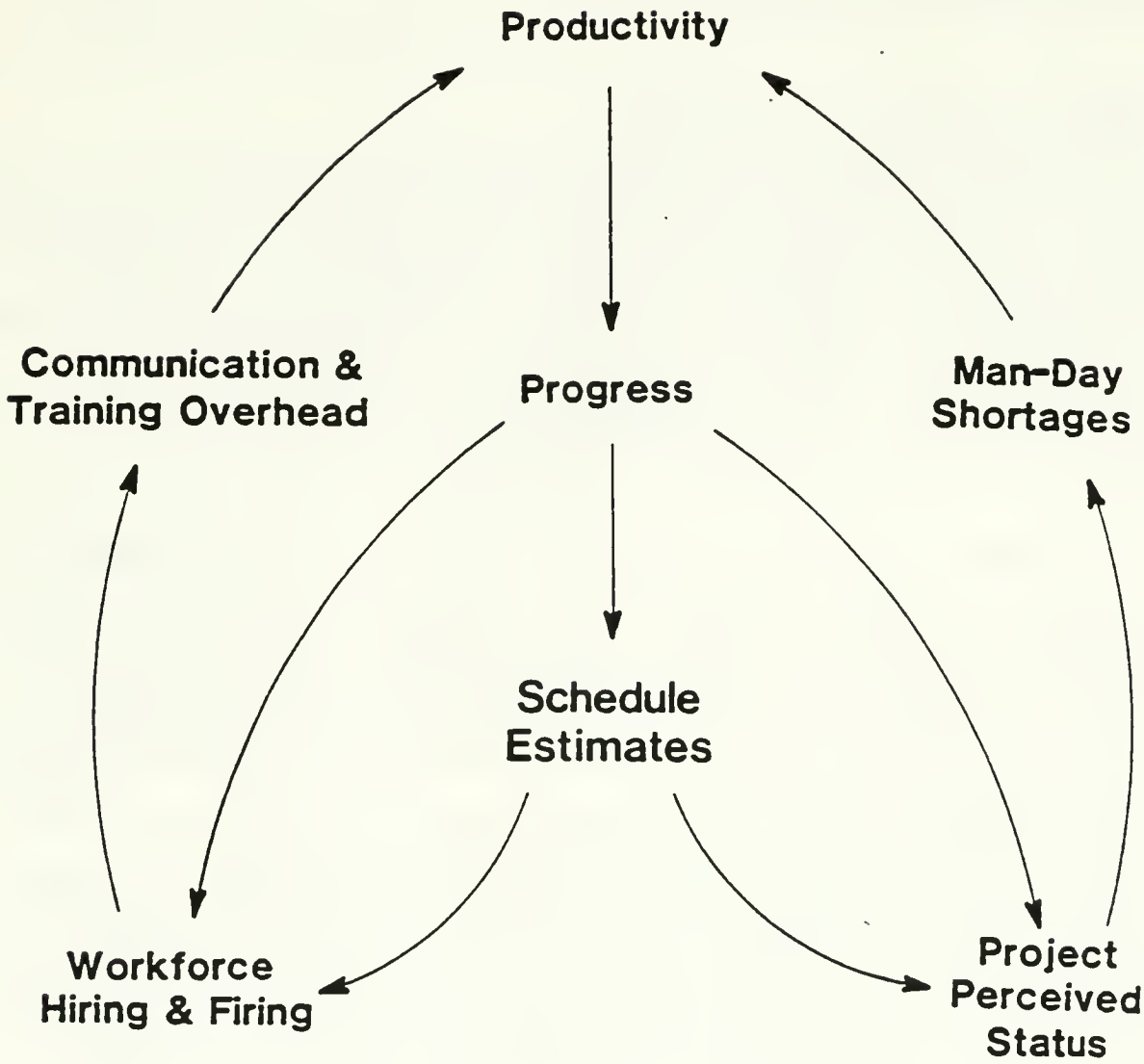
In an analogous manner, by imposing different estimates on a software project we would, in a real sense, be creating different projects. In the next section we will explain how.

Feedback Impact of Project Estimates:

Research findings clearly indicate that the decisions that people make in project situations, and the actions they choose to take are significantly influenced by the pressures and perceptions produced by the project's schedule (Brooks, 1978). Such schedule influences are depicted in the causal loop diagram of Figure 1 [from (Abdel-Hamid, 1984)].

Schedules have a direct influence on the hiring and firing decisions throughout the life of a software project. In TRW's COCOMO model (Boehm, 1981), for example, the project's average staff size can be determined by dividing the man-days estimate (MD) by the development time estimate (TDEV). Thus, for example, a tight time schedule (i.e., a low TDEV value) means a larger work-force. Also, scheduling can dramatically change the manpower loading throughout the life of a project. For example, the workforce 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 workforce level, a project's schedule also affects productivity, as illustrated in Figure 1. This happens because a higher workforce level, for example, means more communication and training overhead, which in turn affects productivity negatively.



Feedback Impact of Schedule Estimates

Figure 1

Productivity is also influenced by the presence of any man-day shortages. For example, if the project is perceived to be behind schedule, i.e., when the total effort still needed to complete the project is perceived to be greater than the total effort actually remaining in the project's budget, software developers tend to work harder, i.e., allocate more man-hours to the project, in an attempt to compensate for the perceived shortage and to bring the project back on schedule. Such man-day shortages are, obviously, more prone to occur when the project is initially under-estimated. Conversely, if project management initially over-estimates the project, man-day "excesses" could arise, and as a result the project would be perceived to be ahead of schedule, i.e., 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." (Boehm, 1981). Which, of course, means that they become less productive.

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

Identifying the feedback relationships through which project estimation influences project behavior is one thing, and discerning the dynamic implications of such interactions on the total system is another. Paraphrasing Richardson and Pugh (1981),

> The behavior of systems of interconnected feedback loops often confounds intuition and analysis, even though the dynamic implications of isolated loops may be reasonably obvious.

One option that might be suggested, is to conduct a controlled experiment, whereby the 64,000 DSI software project is conducted twice under exactly the same conditions, except that in one case it would be initiated with a 2,359 man-day estimate (i.e., on the basis of method "A"), and in the second case a 5,900 man-day estimate would be used (i.e., on the basis of method "B"). While theoretically possible, such an option is usually infeasible from a practical point of view because of its high cost, both in terms of 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 the elements of the model, not reality, but this type of simulation has been found helpful in understandinng the behavior of complex social systems (Richardson and Pugh, 1981).

A Systems Dynamics Computer Model of Software Project Management

The research findings reported in this paper are based on a doctoral thesis research effort, at MIT's Sloan School of Management, to study the dynamics of software development (Abdel-Hamid, 1984). A major outcome of the research was a series of industry interviews and literature study culminating in the development of a system dynamics simulation model of software

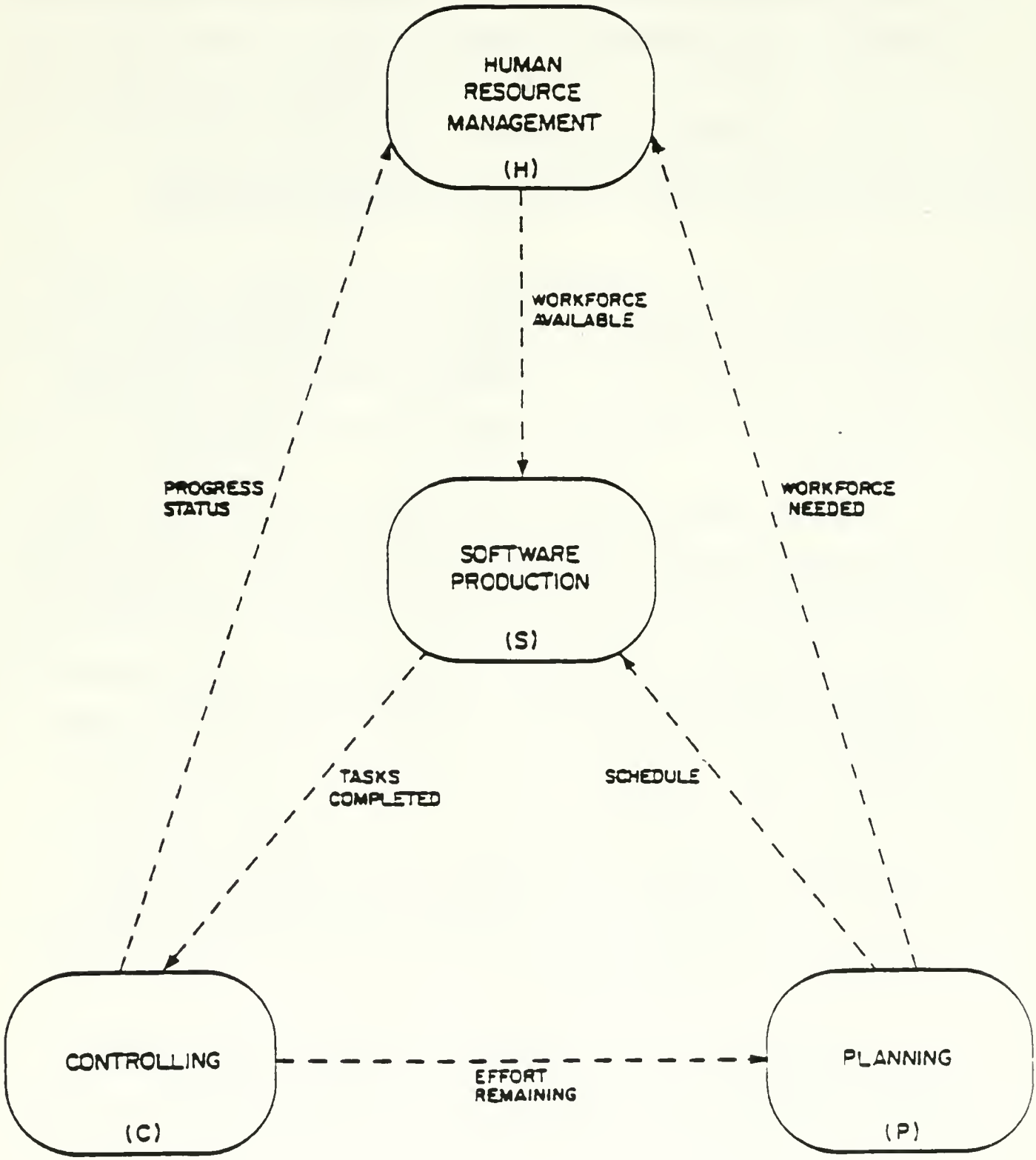
development project management. This structural model served several research objectives, one of which was to serve as a laboratory vehicle for conducting experimentation in the area of software estimation, the topic of this paper. A major advantage of such a structural model is that it can be used to predict the effect of changes to the development process (Barbacci et al, 1985). Our objective in this section is to provide 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. The figure also illustrates some of the interrelatedness of the four subsystems.

The Human Resource Management Subsystem captures the hiring, training, assimilation, and transfer of the project's human resource. Such actions are not carried out in a vacuum, but, 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 "workforce needed" to complete the project on a planned completion date.

Similarly, the "workforce 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. The development activity comprises



OVERVIEW OF SOFTWARE DEVELOPMENT PROJECT MANAGEMENT MODEL

Figure 2

both the design and coding of the software. As the software is developed, it is also reviewed, e.g., using structured-walkthroughs, to detect any design/coding errors. Errors detected through such quality assurance activities are then reworked. Not all errors get detected and reworked, however. Some "escape" detection until the end of development, i.e., until the testing phase.

As progress is made, it is reported. A comparison of where the project is versus where it should be (according to plan) is a control-type activity captured within the Controlling Subsystem. Once an assessment of the project's status is made (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, and then those estimates are revised, when necessary, throughout the project's life. For example, to handle a project that is perceived to be behind schedule, plans can be revised to (among other things) hire more people, extend the schedule, or do a little of both.

The above overview highlights the structure of the model used. A full description of the model and of the validation experiments performed on it are provided in other reports [(Abdel-Hamid, 1984), and (Abdel-Hamid and Madnick, 1986)].

A Simulation Experiment on Software Estimation Accuracy:

The scenario for our simulation experiment is based on a real-life software development environment in one organization, a major mini-computer manufacturer, which was involved in this research effort. In the particular organization, project managers were rewarded on how close their project estimates were to their actual project results. The estimation procedure that they informally used was as follows:

1. Use Basic COCOMO to estimate the number of man-days (MD). That is, use

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

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

2. Multiply this estimate by a safety factor (the safety factor ranged from 25% to 50%) and add it to MD, i.e., MD' = (1 + 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 the estimates are made) the real size of the project is often not known (Boehm, 1981). It is also important, for purposes of this paper, to note that COCOMO is only used as an example of a schedule estimation tool, the structural model developed is not tied to any particular schedule estimation technique.

The Safety Factor Philosophy is not, in any way, unique to this one organization. For example, in a study of the software cost estimation process at the Electronics Systems Division of the Air Force Systems Command, Devenny (1976) 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% with a mean of 18%. And as was the case in the organization we studied, the policy was an informal one: "...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" (Devenny, 1976).

To test the efficacy of various Safety Factor policies, we ran a number of simulations on a prototype software project which we will call project EXAMPLE. Project EXAMPLE's actual size is 64,000 DSI. However, at its initiation, it was incorrectly perceived as being 42.88 KDSI in size (i.e., 33% smaller than 64 KDSI). This incorrectly perceived project size (of 42.88 KDSI) was then the input used in COCOMO's estimation equations. The Basic COCOMO estimate for man-days (without any safety factor) was:

$$MD = 2.4 * 19 * (42.88)^{1.05} = 2,359 \text{ man-days}$$

We can see that this estimate corresponds to the method "A" estimate presumed at the beginning of this paper.

We experimented with Safety Factor values ranging from 0 (the base run) to 100%. For example, for a Safety Factor of 50%, the following estimates would be used:

1. Calculate MD' from MD

   MD' = MD * (1+Safety Factor/100)
   = MD * 1.5 = 3,538.5 man-days

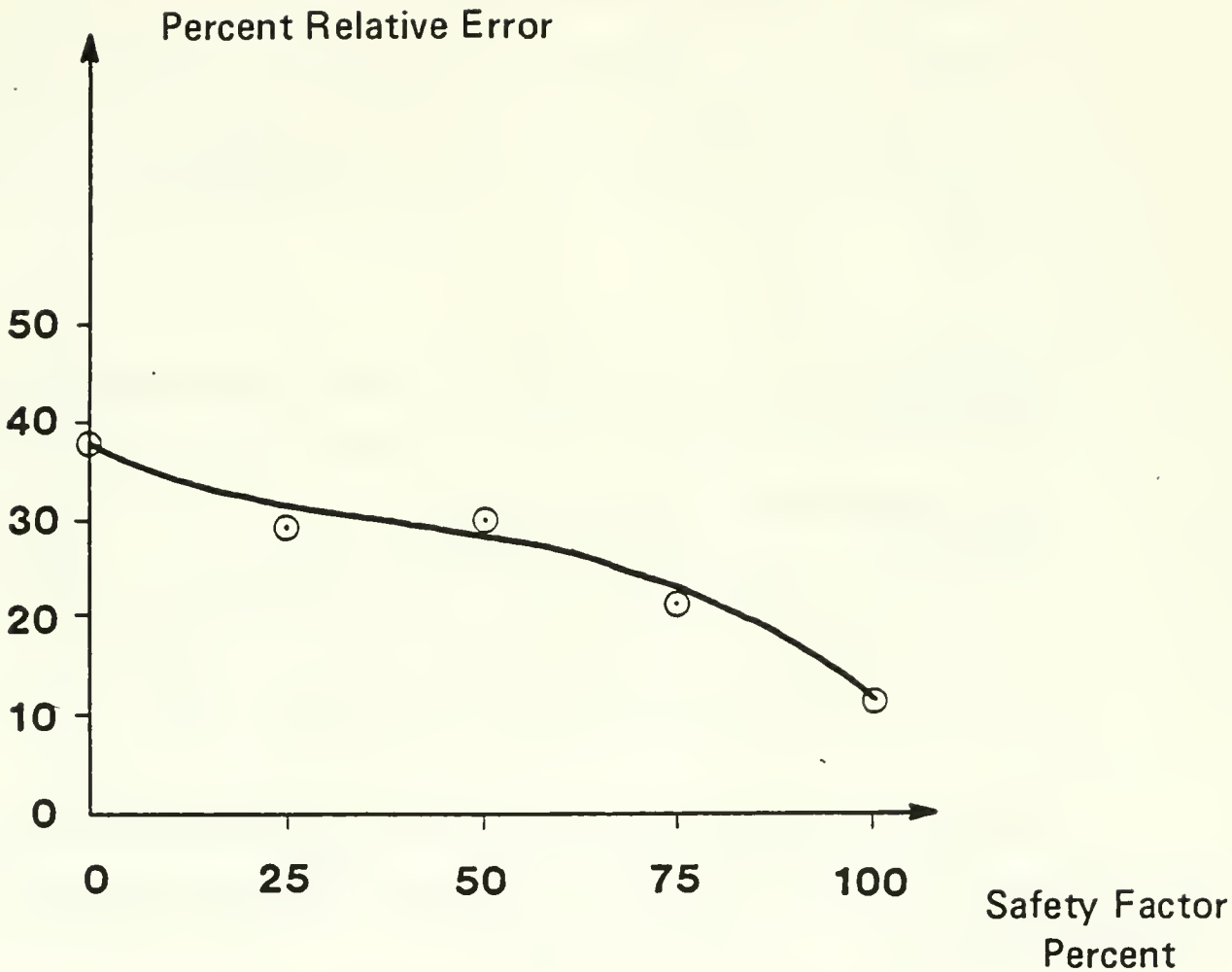
2. Calculate TDEV

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

The results of these simulations are exhibited in Figures 3 through 6.

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 particular, in the 25-50% range (which is what was used in the organization) the estimation error drops from being approximately 40% in the base run, to values in the upper twenties. In fact, Figure 3 suggests that by using a Safety Factor in the 25-50% range, the project managers might not be going far enough, since a 100% Safety Factor, for example, would drop the estimation error down to a "more rewarding" 12%.

The rational, or the justification, for using a Safety Factor (as also observed in our interviews with software development managers) is based on the following set of assumptions:



Percent Relative Error in Estimating Actual Man-Days

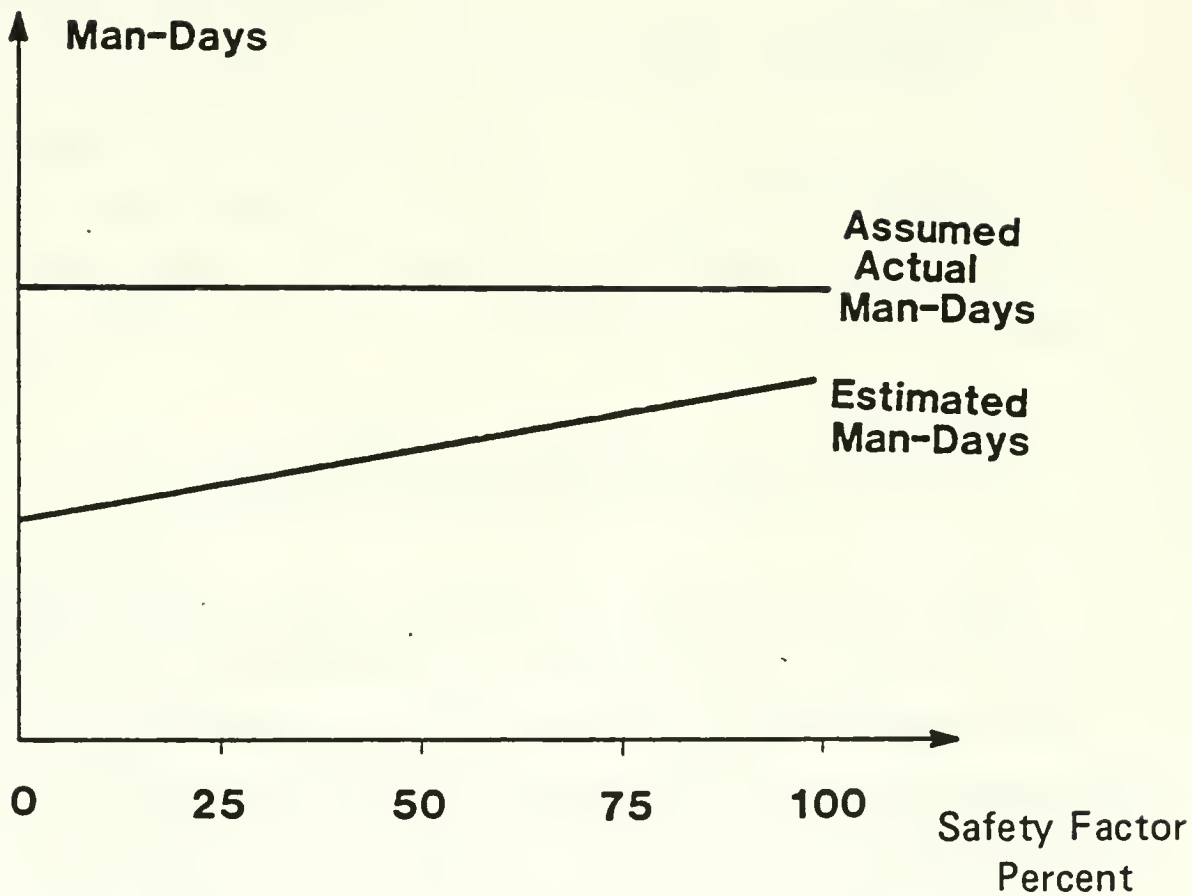
Figure 3

1. Past experiences indicate a strong bias on the part of software developers to underestimate the scope of a software project [(Devenny, 1976) and (DeMarco, 1982)].

2. "(One) might think that a bias would be the easiest kind of estimating problem to rectify, since it involves an error that is always in the same direction ... (But biases) are, almost by definition, invisible ... the same psychological mechanism (e.g., optimism of software developers), that creates the bias works to conceal it" (DeMarco, 1982).

3. To rectify this bias on the part of software developers (e.g., system analysts and designers), project management uses a Safety Factor. When the project manager "...adds a contingency factor (25%? 50% 100%) he is, in effect, saying that: 'much more is going to happen that I don't know about, so I'll estimate the rest as a percentage of that which I do know something about'" (Pietrasanta, 1968).

In other words, the assumption is that the Safety Factor is simply a mechanism to bring the initial man-days estimate closer to the project's true size in man-days ... as shown in Figure 4.



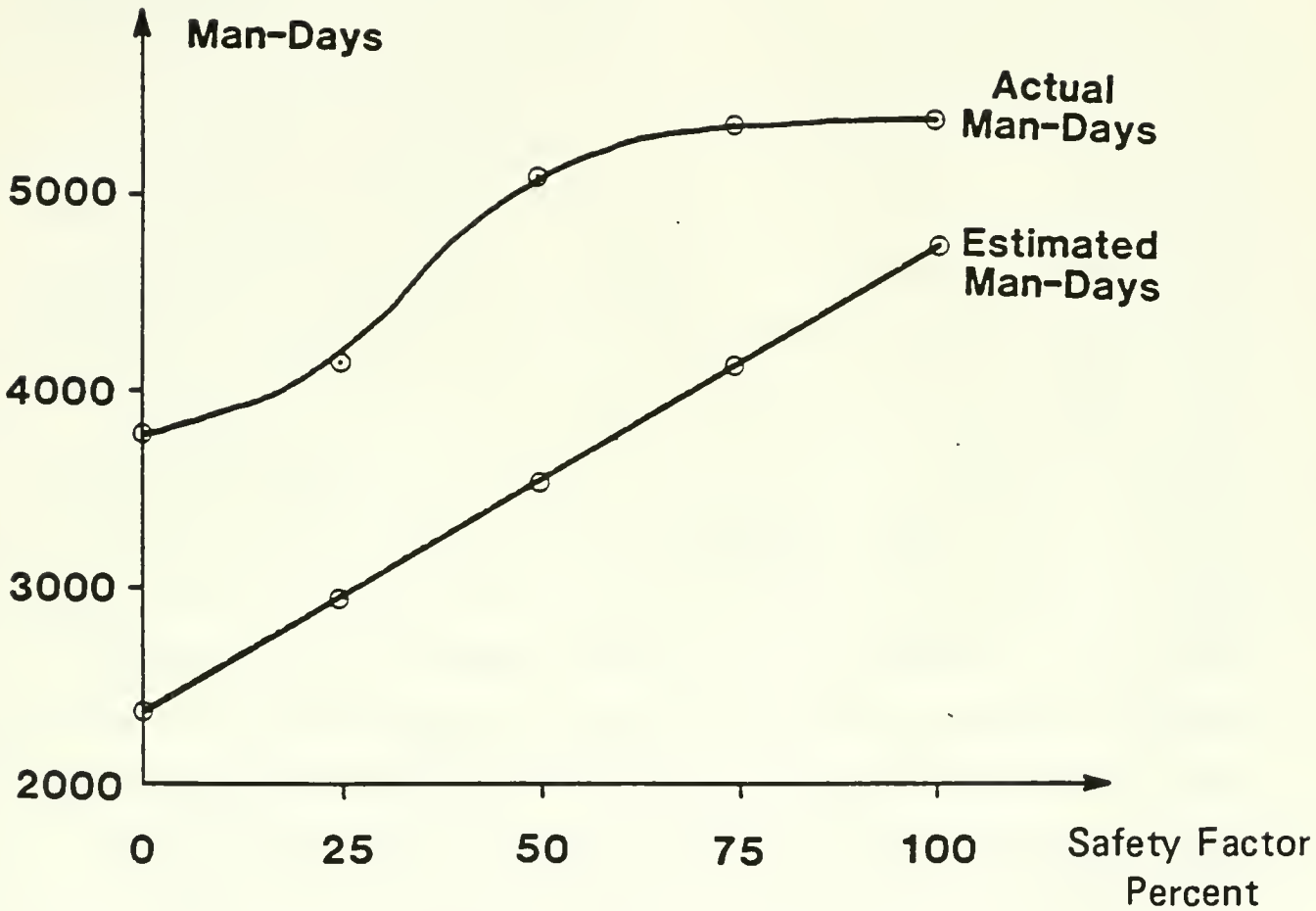
Comparison of Assumed Actual Man-Days with Estimated Man-Days

Figure 4

Notice that such an assumption cannot be contested solely on the basis of Figure 3 which provides only part of the story. A more complete picture is provided by Figure 5, where the model was used to calculate the actual man-days that would have been consumed by the project EXAMPLE, when different Safety Factors were applied to its initial estimate. The assumption of Figure 4 is obviously invalidated. As higher Safety Factors are used, leading to more and more generous initial man-day allocations, the actual amount of man-days consumed, does not remain at some inherently-defined value. For example, in the base run, project EXAMPLE would be initiated with a man-day estimate of 2,359 man-days and would end up consuming 3,795 man-days. When a Safety Factor of 50% is used, i.e., leading to a 3,538 man-day initial estimate, EXAMPLE ends up consuming, not 3,795 man-days, but 5,080 man-days. To reiterate a point made earlier:

A different estimate creates a different project.

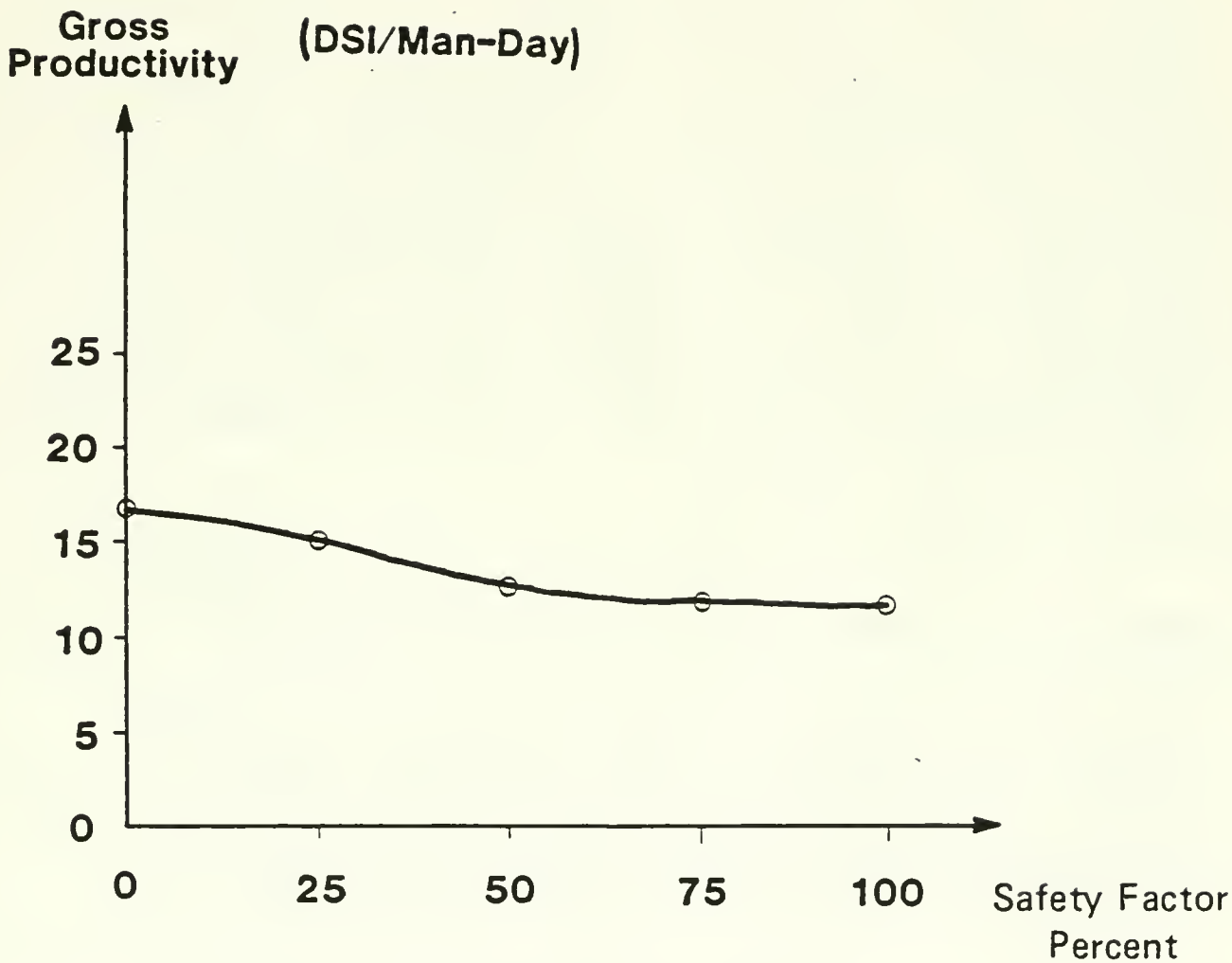
The reason this happens, is that the project's initial estimates create (as was explained earlier) pressures and perceptions that affect how people behave on the project. In particular, an overestimate of the project's man-days can lead to a larger buildup of the project's workforce, leading to higher communication and training overheads, which in turn affect productivity negatively. In addition, when a project is overestimated, it often leads to an expansion of the project members' discretionary activities (e.g., non-project communication, personal activities, etc.), leading to further reductions in productivity.



Comparison of Actual Man-Days with Estimated Man-Days

Figure 5

Figure 6 is a plot of "Gross Productivity," which is defined as the project size in DSI (i.e., 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 a 100% Safety Factor is used. Notice that 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 (i.e., in the base run) to a relatively small value (e.g., 25%) most of the man-day excesses that result (and which in this case would be moderate) would be absorbed by the employees in the form of less overworking (e.g., less days in which employees work longer-than-usual hours) and/or more discretionary time. Remember, in the base case (no safety factor used), backlogs are experienced as project EXAMPLE consumes more man-days (3,795) than was budgeted (2,359). When a small Safety Factor is used, 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 largely absorbed in the form of reasonably expanded discretionary activities. Which, of course, means that the project team becomes less productive. However, there is a limit 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 workforce, its schedule, or both (Abdel-Hamid, 1984). Thus, as the Safety Factor increases to larger and larger values, losses in productivity due to the expansion of the slack time activities decrease, leading to lower drops in Gross Productivity.



Gross Productivity

Figure 6

Revisit Method "A" and Method "B" Comparison:

We are now in a position to answer the question posited at the beginning of this discussion. The situation concerned the 64,000 DSI project, which is in fact our own project EXAMPLE, and a comparison of two estimation methods. Method "A" produces a 2,359 man-day estimate. It is, in other words, the estimate used in the base run. Since project EXAMPLE ended up actually consuming 3,795 man-days, the percent of relative absolute error in estimating man-days is 38%. We then questioned whether another estimation method "B," which produces a 5,900 man-day estimate for project EXAMPLE (i.e., an estimate that is 55% higher than base run EXAMPLE's man-day expenditures of 3,795), would have provided a less accurate estimate of the project's man-days, had it been used instead of method A.

Notice that method "B's" estimate of 5,900 man-days is exactly 150% higher than "A's" 2,359 estimate i.e., method "B" is equivalent to a "Safety Factor Policy " in which the Safety Factor is set to 150%. To check the behavior of project EXAMPLE had it been estimated using Method "B," we re-ran the model with a man-days estimate (MD) equal to 5,900. The results of the run, together with those of the base case are tabulated below:

	Method "A" (Base Run)	Method "B"
$MD_{ESTIMATE}$	2,359	5,900
MD_{ACTUAL}	3,795	5,412
% Error	38%	9%

The results are quite interesting. Whereas method "A" had a 38% error, method "B" with only a 9% error turns out to be, in fact, a more accurate estimator. However, the improved accuracy of using method "B" is attained at a high cost. The project turns out consuming 43% man-days more!

In terms of the mini-computer manufacturer we studied for this experiment, the message is clear. The "Safety Factor Policy" does achieve its intended objective, namely, producing relatively more accurate estimates. However, the organization is paying dearly for this. As Figure 5 indicates, a Safety Factor in the 25-50% range results in a 15-35% increase in the project's cost in terms of man-days. Taking an extreme case, if a development manager was required to complete the project within 10% of estimate, method "B" would be a very appropriate approach for that manager. But, this would not necessarily be a good economical approach for the company (since costs would be 43% higher than method "A").

Conclusion:

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

1. A different estimate creates a different project. The important implication that follows from this is that both the project manager as well as the student of software estimation should reject the notion that a software estimation tool can be adequately judged on the basis of how accurately it can estimate historical projects.

2. A more accurate estimate is not necessarily a better estimate. An estimation method should not be judged only on how accurate it is, but in addition it should be judged on how costly the projects it "creates" are.

Acknowledgements:

We acknowledge the contribution of each and every individual in the organizations that provided perspectives and data to this overall research effort. We thank the reviewers whose suggestions have helped improve the readibility of this paper. Work reported herein was supported, in part, by research grant NAGW-448 From NASA.

References

1. Abdel-Hamid, T. K. "The Dynamics of Software Development Project Management: An Integrative System Dynamics Perspective." NTIS, N84 16824, (1984).

2. Abdel-Hamid, T. K. and Madnick, S. E. Software Project Management Englewood Cliffs, New Jersey: Prentice-Hall, Inc., to be published 1986.

3. Barbacci, Mario R., Habermann, A. Nico, and Shaw, Mary, "The Software Engineering Institute: Bridging Practice and Potential." IEEE Software, November, 1985, 4-21.

4. Boehm, B.W. Software Engineering Economics. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1981.

5. Brooks, F.P. The Mythical Man-Month. Reading, MA: Addison-Wesley Publishing Co., 1978.

6. DeMarco, T. Controlling Software Projects. New York: Yourdon Press, Inc., 1982.

7. Devenny, T.J. "An Exploration Study of Software Cost Estimating at the Electronics Systems Division." NTIS, U.S. Department of Commerce, (July, 1976).

8. Farquhar, J. A. A Preliminary Inquiry into the Software Estimation Process. Technical Report, AD F12 052, Defense Documentation Center, Alexandria, Va., (August, 1970).

9. Koolhass, Z. Organization Dissonance and Change. New York: John Wiley & Sons, Inc., 1982.

10. Mohanty, S.N. "Software Cost Estimation: Present and Future." Software-Practice and Experience, Vol, 11, (1981), 103-121.

11. Pietrasanta, A.M. "Managing the Economics of Computer Programming." Proceedings of the ACM National Conference, 1968, 341-346.

12. Richardson, G.P. and Pugh, G. L. III. Introduction to System Dynamics Modeling with Dynamo. Cambridge, MA: The MIT Press, 1981.