1990

On the Utility of Historical Project Statistics for Cost and Schedule Estimation: Results from a Simulation-based Case Study

Abdel-Hamid, Tarek K.

Science Direct

Abdel-Hamid, Tarek K. "On the utility of historical project statistics for cost and schedule estimation: results from a simulation-based case study."Journal of Systems and Software13.1 (1990): 71-82.
http://hdl.handle.net/10945/68470

On the Utility of Historical Project Statistics for Cost and Schedule Estimation: Results from a Simulation-based Case Study

Tarek K. Abdel-Hamid

Department of Administrative Sciences, Naval Postgraduate School, Monterey, California

Estimating the duration and cost of software projects has traditionally been, and continues to be, fraught with peril. This is in spite of the fact that over the last decade a large number of quantitative software estimation models have been developed. Our objective in this article is to challenge two fundamental assumptions that underlie research practices in the area of software estimation, which may be directly contributing to the industry's poor track record to date. Both concern the "fitness" of raw historical project statistics for calibrating and evaluating (new) estimation models.

A system dynamics model of the software development process is developed and used as the experimentation vehicle for this study. An overview of the model's structure is presented, followed by a discussion of the two experiments conducted and their results. In the first, we demonstrate why it is inadequate to assess the accuracy of (new) estimation tools simply on the basis of how accurately they replicate old projects. Second, we show why raw historical project results do not necessarily constitute the most "preferred" and reliable benchmark for future estimation.

1. INTRODUCTION

Estimation of software cost and schedule is an essential foundation for software project planning and control. Without accurate cost and schedule estimates,

> ··· 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 ··· [1]

Since the early 1950s, software development practitioners as well as researchers have been trying to develop algorithmic models to accurately estimate software costs and schedules. The earliest attempts were simple rules of thumb, such as "On a large project, each software performer will provide an average of one checked-out instruction per man-hour" [2]. More recently, organizations began collecting quantitative data during software development that characterize both the product under development (e.g., software complexity) as well as the software development process (e.g., effort and schedule). This provided a basis for the development of a large number of quantitative estimation models over the last decade [e.g., 3–12].

Still, the accuracy of such quantitative tools has proven inadequate [13–15]. As a result, many software development organizations do not seem to trust any of the available quantitative models. A study of 30 organizations showed that the models were used only to "check manual estimates" [16].

Our objective in this article is to challenge two fundamental assumptions that underlie research practices in the area of software estimation and that are may be directly contributing to the industry's poor track record to date. Both concern the "fitness" of *raw* historical project statistics for calibrating and evaluating estimation models.

Consider, as an example, the case of a NASA software project that involved the development of a software system for controlling a NASA satellite. The system's size was initially estimated to be 16,000 delivered source instructions (DSI), and the cost and schedule were estimated to be 1,100 man-days and 320 working days, respectively. Upon completion, the project's actual results were as follows:

Project size	24,400 DSI
Development cost	2,200 man-days
Completion time	380 working days

Address correspondence to Tarek K. Abdel-Hamid, Department of Administrative Sciences, Naval Postgraduate School, Monterey, CA 93943.

0164-1212/90/$3.50

The above project statistics were then directly incorporated into a database of historical project results that was developed to support two kinds of activities—first, to support the development, calibration, and fine-tuning of software estimation tools. The underlying assumption here is that project results, such as the ones described earlier, constitute the most "preferred" and reliable benchmark for future estimation purposes ··· after all they are *actual* values.

Second, completed project results are used in the ex-post evaluation of estimation models. That is, the accuracy of estimation models is assessed on the basis of how closely they replicate the results of old projects [17, 18]. For example, the accuracy of a new estimation tool can be assessed by applying it to the earlier project and calculating its resultant percent relative error. Thus, if some tool generates say 3,500 and 237 estimates for the cost and duration of the project above, the percent relative error in estimating the latter would be:

$$\text{Percent Relative Error} = \frac{100^* \,|\, \text{Actual} - \text{Estimate} \,|}{\text{Actual}}$$

$$= \frac{100^* \,|\, 380\text{-}237 \,|}{380}$$

$$= 37.6$$

Notice the rather fundamental assumption that underlies such a practice, namely, that the project's completion time and cost *will remain to be* 380 days and 2,200 man-days, respectively. That is, it is assumed in the above calculation that a project's final cost and schedule values are independent of its initial estimation values!

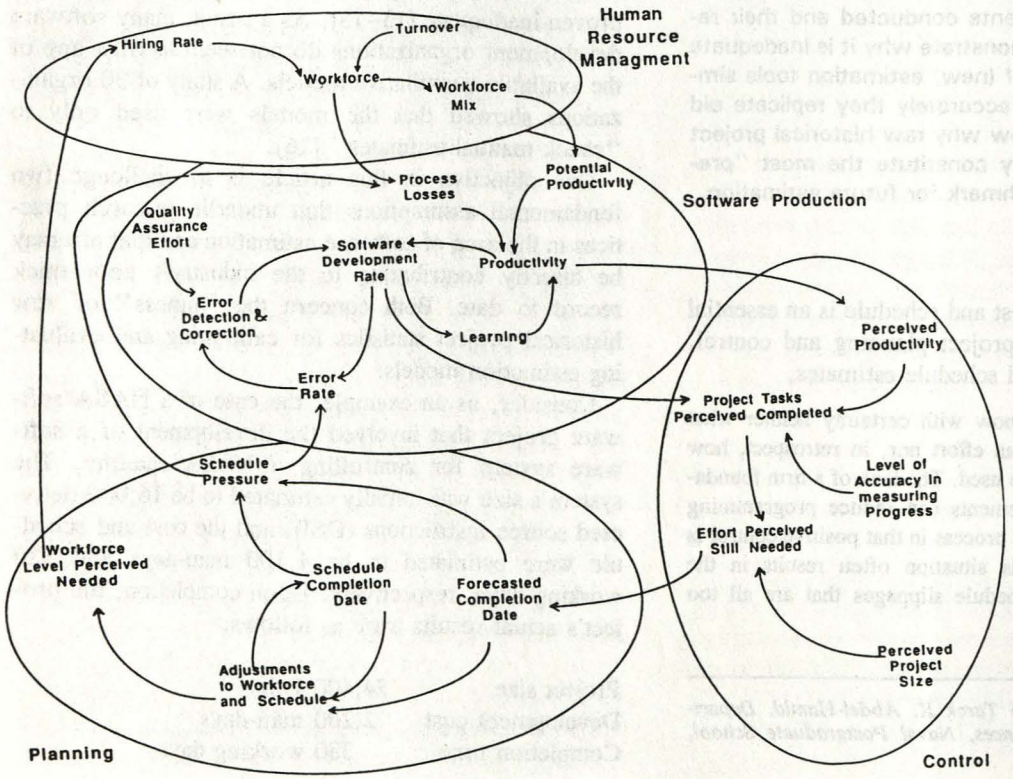
Both of the above assumptions are flawed. In the remainder of this article, we explain why they are flawed and discuss the implications for the practice of software cost and schedule estimation. But first, we discuss in the next section the system-dynamics-based simulation approach employed in this study. An overview of the model's structure is first presented, followed by a full discussion of the experiments conducted and their results.

2. A SYSTEM DYNAMICS MODEL OF SOFTWARE DEVELOPMENT

Our work on software project estimation is part of a larger project to study the dynamics of the entire software-development process. A major part of this effort is the development of a comprehensive system-dynamics model of software development.

The model was based on a field study of project managers in five organizations. Figure 1 shows a high-level view of the model's four subsystems: human

Figure 1



resource management, software production, control, and planning, and some of the relations between them. The actual model is very detailed and contains more than 100 causal links; a full description of the model's structure, its mathematical formulation, and its validation is published elsewhere [19–21].

2.1 Human Resource Management

This subsystem captures the hiring, training, assimilation, and transfer of the human resource. The project's total work force is segregated into different types of employees (newly hired and experienced). We make this distinction because new team members are usually less productive than veterans [22]. This segregation also allows us to capture the training process to assimilate new members. The veterans usually train the newcomers, both technically and socially [23, 24]. This is important, because this training can significantly affect a project's progress by reducing the veteran's productivity.

In deciding how big a work force they need, project managers typically consider several factors. One, of course, is the project's scheduled completion date. Another is the work force's stability, so managers try to predict project employment time for new members before they are hired. In general, the relative weight managers give to stability versus completion date changes as the project progresses.

2.2 Software Production

This subsystem models development; it does not include the operation and maintenance phases. The development phases included are designing, coding, and testing but not the initial requirements definition phase. We chose not to include requirements definition for two reasons. First, our focus is on the indigenous development organization: project managers and developers, and how their policies, decisions, and actions affect development. In many organizations, defining user requirements is not completely within the control of this group. Second, "Analysis to determine requirements is distinguished as an activity apart from software development. Technically, the product of analysis is nonprocedural (i.e., the focus is functional)" [25].

As software is developed, it is reviewed to detect any errors, for example, using quality assurance activities such as structured walkthroughs. Errors detected through such activities are reworked. Not all software errors are detected during development, however, since some escape detection until the testing phase.

The software production subsystem models productivity and its determinants in great detail. Productivity is defined as potential productivity minus the loss from faulty processes. Potential productivity is "the maximum level of productivity that can occur when an individual or group ··· makes the best possible use of its resources" [26], and is a function of the nature of the task and the group's resources. Losses from faulty processes are losses in productivity from things like communication and coordination overheads and low motivation.

2.3 Control Subsystem

In all organizations, decisions are based on the information available to the decision maker. Often, this information is inaccurate. Apparent conditions may be far removed from those actually encountered, depending on information flows, time lag, and distortion. Thus, system dynamicists go to great lengths to distinguish between actual and perceived model variables [27].

Progress rate is a good example of a variable that is difficult to assess during the project. Because software is basically an intangible product during most of the development, it is difficult to measure things like programming performance and intermediate work [28].

How can you measure progress? Our own field studies corroborated published reports in which progress, especially in the earlier phases of development, is typically measured by the rate of resource expenditure rather than by accomplishments [29]. Baber [30] explains:

> It is essentially impossible for the programmers to estimate the fraction of the program completed. What is 45% of a program? Worse yet, what is 45% of three programs? How is he to guess whether a program is 40% or 50% complete? The easiest way for the programmer to estimate such a figure is to divide the amount of time actually spent on the task to date by the time budgeted for that task. Only when the program is almost finished or when the allocated time budget is almost used up will he be able to recognize that the calculated figure is wrong.

When you measure progress this way, status reports become nothing more than an echo of the original plan. As the project advances toward its final stages, work accomplishments become relatively more visible, and project members better perceive how productive the work force has actually been.

2.4 Planning Subsystem

In the planning subsystem, you make project estimates, revising them as the project progresses. For example, when a project is behind schedule, you can revise the plan to hire more people, extend the schedule, or both.

Figure 2 shows a detailed causal-loop structure of the

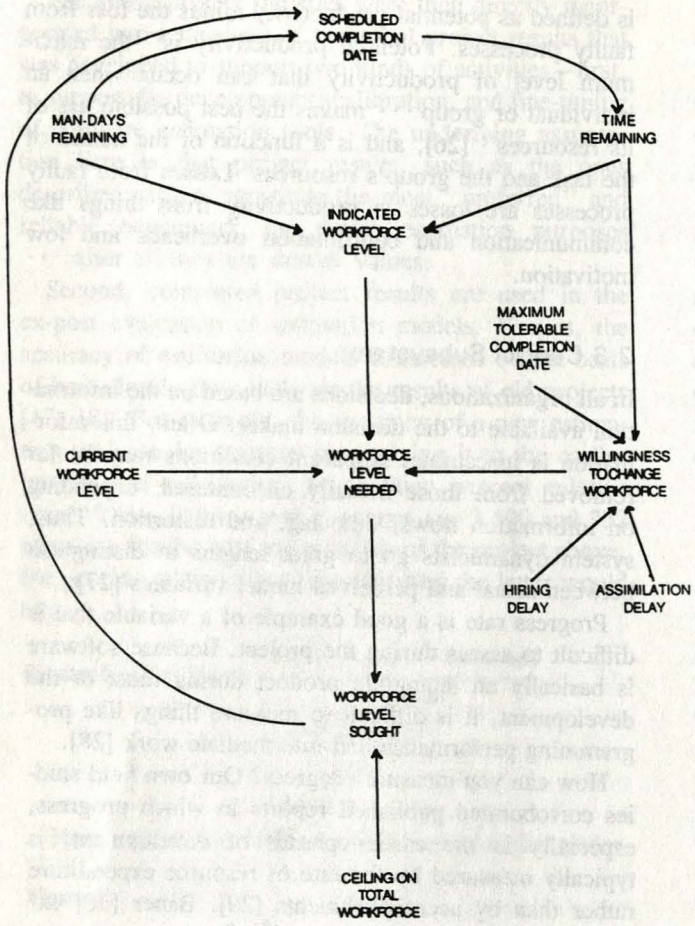


Figure 2. The planning subsystem.

adjustments to work force and schedule. By dividing the value of man-days remaining at any point in the project by the time remaining, a manager can determine the *indicated* work-force level, which is the work force needed to complete the project on time.

Hiring decisions are not made solely on the basis of scheduling requirements. Managers must also consider the training requirements and the work force's stability. In general, the relative weighting between the desire for work-force stability and the desire to complete the project on time is not static; it changes through the project's life.

Although management determines the work-force level needed to complete the project, this level does not necessarily translate into the actual hiring goal (the work-force level sought in Figure 2). The hiring goal is constrained by the ceiling on new hires. This ceiling represents the highest work-force level management believes can be adequately handled by its experienced project members.

Thus, three factors–scheduled completion time, work-force stability, and training requirements—affect the work-force level.

THE DE-A SOFTWARE PROJECT: A CASE STUDY

As part of model validation, a case-study was conducted at NASA to test the model's accuracy in replicating the dynamic behavior of a real software project, namely, NASA's DE-A project. (NASA was *not* one of the five organizations studied during model development.) The DE-A project, conducted by the Systems Development Section of the Goddard Space Flight Center (GSFC) at Greenbelt, Maryland was to design, implement, and test a software system for processing telemetry data and providing attitude determination and control for the DE-A satellite. The FORTRAN system runs on IBM system 360/95 and system 360/75 mainframes.

At the start of the project, the estimates for system size, total development effort, and schedule were 16,000 source instructions, 1,100 man-days, and 320 working days, respectively. On completion, the DE-A project had delivered 24,400 source instructions in 2,200 man-days and 380 working days.

Figure 3 shows the model's simulation run of the

J. SYSTEMS SOFTWARE
1990; 13:71-82

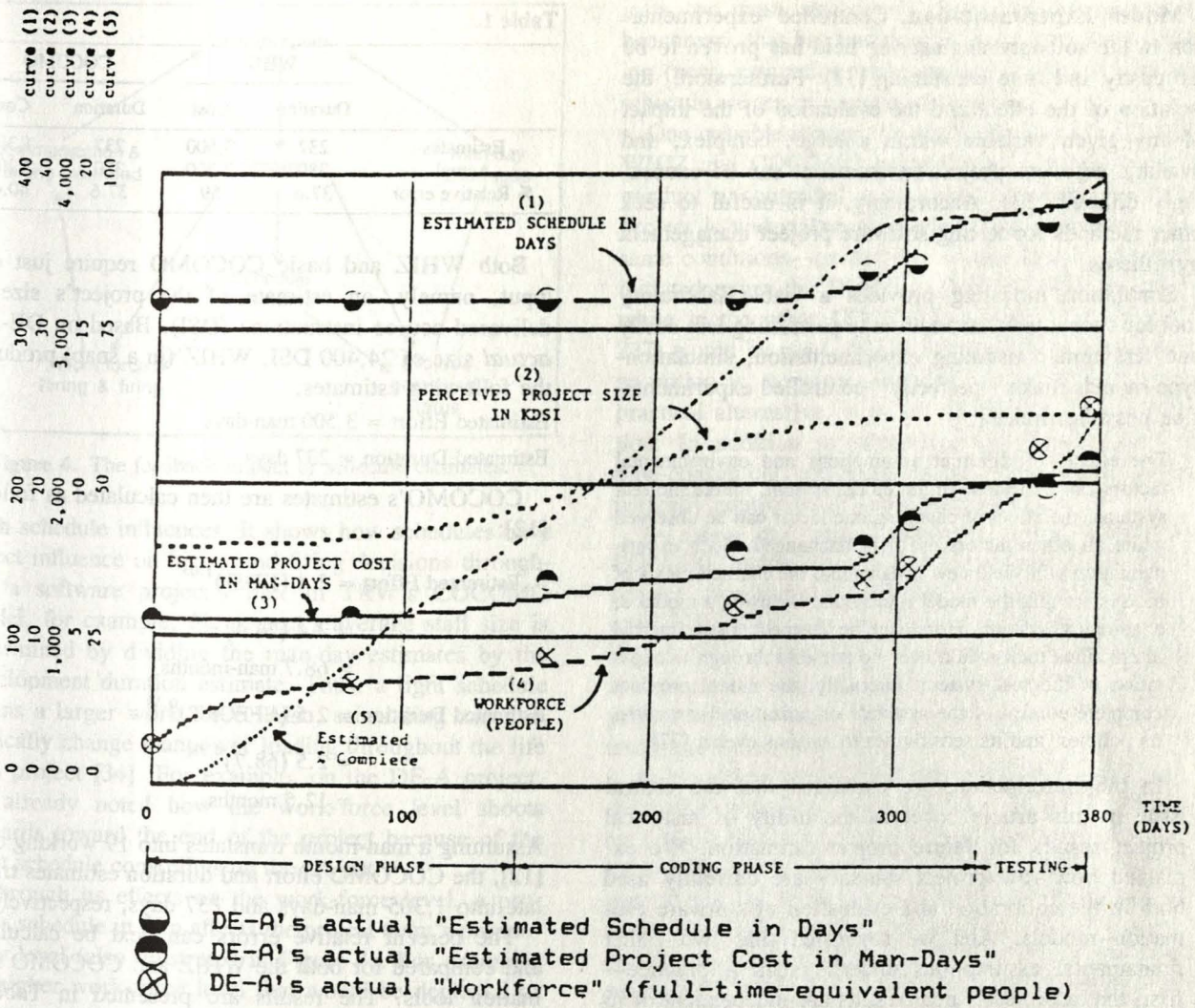


Figure 3. Model simulation of the DE-A project.

DE-A software project. As shown, the model accurately replicated the project's actual behavior. The figure illustrates that DE-A's management held to the project's estimated schedule in days during most of the project's design and coding phases, despite a gradual increase in the perceived project size. To maintain the schedule, management added to the project's work force. This behavior is not atypical. It arises, according to DeMarco (1982) from political considerations:

> Once an original estimate is made, it's all too tempting to pass up subsequent opportunities to estimate by simply sticking with your previous numbers. This often happens even when you know your old estimates are substantially off. There are a few different possible explanations for this effect: It's too early to show slip . . . If I re-estimate now, I risk having to do it again later (and looking bad twice) . . . As you can see, all such reasons are political in nature.

The DE-A project's work force pattern, on the other hand, does *not* conform to the staffing pattern typically portrayed in the literature where the work-force level rises, peaks, and then drops back to lower levels as the project nears the system testing phase (Boehm, 1981). Instead, the work force level rises steadily because NASA tied the launch of the satellite to the completion of the software. All software had to be accepted and frozen 90 days before launch and no serious schedule slippages were tolerated.

Therefore, as the project approached this maximum tolerable completion date, pressures developed that overrode considerations of work-force stability. Management would pay any price to avoid overshooting the 90-day-before-launch date. This translated, as Figure 3 indicates, into a hiring binge late in the life cycle. (In Abdel-Hamid [31], we investigate whether such a staffing policy did or did not contribute to the project's late completion.)

Model Experimentation. Controlled experimentation in the software engineering field has proven to be too costly and time consuming [32]. Furthermore, the isolation of the effect and the evaluation of the impact of any given variable within a large, complex, and dynamic software project environment can be exceedingly difficult [33]. Accordingly, it is useful to seek other methods for testing software project management hypotheses.

Simulation modeling provides a viable laboratory tool for such a task. In addition to permitting less costly and less time-consuming experimentation, simulation-type models make "perfectly" controlled experimentation possible. Indeed:

> The effects of different assumptions and environmental factors can be tested. In the model system, unlike the real systems, the effect of changing one factor can be observed while all other factors are held unchanged. Such experimentation will yield new insights into the characteristics of the system that the model represents. By using a model of a complex system, more can be learned about internal interactions than would ever be possible through manipulation of the real system. Internally, the model provides complete control of the system's organizational structure, its policies, and its sensitivities to various events [27].

In the introduction, we explained that the central issue in this article concerns the utility of historical project results for future project estimation. We explained how raw project statistics are currently used both in the calibration and evaluation of software estimation models. And we suggested that two rather fundamental assumptions underlie such a practice—first, that a project's final results are independent of its initial estimation values, and second, that historical project results constitute the most "preferred" and reliable benchmarks for future estimation purposes.

In the next sections, we will utilize our model as an experimentation vehicle to show why the above two assumptions are flawed. We will also discuss the implications for the practice of software cost and schedule estimation.

EXPERIMENT 1: DIFFERENT ESTIMATES CREATE DIFFERENT PROJECTS

Let us assume that NASA is considering the adoption of one of two proposed estimation tools. The first is a software estimation "magician" called WHIZ, and the second is the basic version of TRW's COCOMO model [12]. To determine which of the two is more suited to the NASA environment, a test is conducted to assess the accuracy of the tools in replicating the results of the DE-A project.

Table 1.

	WHIZ		COCOMO	
	Duration	Cost	Duration	Cost
Estimates	237	3,500	237	1,305
Actual	380	2,200	380	2,200
% Relative error	37.6	59	37.6	40.6

Both WHIZ and basic COCOMO require just one input, namely, an estimate of the project's size in delivered source instructions (DSI). Based on DE-A's *actual* size of 24,400 DSI, WHIZ (in a snap) produces the following estimates:

Estimated Effort = 3,500 man-days

Estimated Duration = 237 days

COCOMO's estimates are then calculated as follows [12]:

$$\text{Estimated Effort} = 2.4\ (\text{KDSI})^{1.05}$$
$$= 2.4\ (24.4)^{1.05}$$
$$= 68.7 \text{ man-months}$$

$$\text{Estimated Duration} = 2.5\ (\text{EFFORT})^{0.38}$$
$$= 2.5\ (68.7)^{0.38}$$
$$= 12.5 \text{ months}$$

Assuming a man-month translates into 19 working days [12], the COCOMO effort and duration estimates translate into 1,305 man-days and 237 days, respectively.

The percent relative errors can next be calculated and compared for both the WHIZ and COCOMO estimation tools. The results are presented in Table 1 below:

We would like to suggest that the above analysis has one basic flaw. It is in assuming that DE-A's final cost and schedule values are independent of the initial estimates used at the start of the project. That is, it is incorrect to assume that the DE-A project's final cost and schedule values under the two new estimation scenarios will remain to be 2,200 man-days and 380 days (as in the base case). The reason is simply this: different initial project estimates do, in a very real sense, "create" different projects. This assertion is supported by two pieces of evidence; first, research findings reported in the literature, and, second, our own experimental results.

Research findings 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 that project schedules produce [28, 34-36]. Figure 4's causal loop diagram depicts

J. SYSTEMS SOFTWARE
1990; 13:71-82

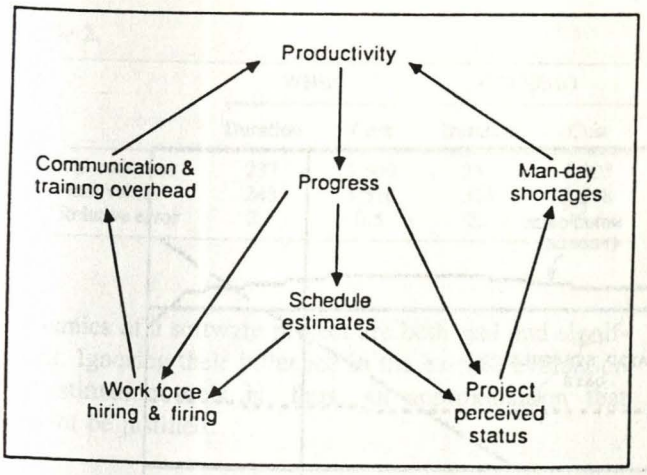


Figure 4. The feedback impact of schedule estimates.

such schedule influences. It shows how schedules have direct influence on hiring and firing decisions throughout a software project's life. In TRW's COCOMO model, for example, the project's average staff size is determined by dividing the man-day estimates by the development duration estimate. Thus, a tight schedule means a larger work force. Also, scheduling can dramatically change manpower loading throughout the life of a project [34]. For example, on the DE-A project, we already noted how the work-force level shoots upwards toward the end of the project because of the strict schedule constraints on the project.

Through its effects on the work-force level, a project's schedule in turn affects the project team's productivity level (also illustrated in Figure 4). For example, the higher work-force level that would be deployed to deliver a project on a tighter schedule often leads to higher communication and training overheads on the project, which in turn leads to a decrease in productivity [12, 34, 37, 38].

In addition, productivity can be influenced by *perceptions*. For example, if a project is perceived to be behind schedule, software developers will tend to work harder in order to bring the project back on schedule [35]. In one empirical study, Boehm [12] reports that team members doubled their effort as schedule pressures mounted prior to major project milestones.

Thus, initial project cost and schedule estimates influence hiring and firing decision, productivity, communication and training overheads, and work intensity. All are critical factors that in turn significantly influence the cost and schedule of software development.

Note that such a "revelation" does introduce a significant complication into the task of evaluating new estimation tools. For once we accept the notion that different initial project estimates create different projects, we must immediately disqualify that convenient benchmark that has traditionally been used for evaluating (new) estimation tools, namely, the *raw* cost and schedule values of completed projects.

One possible strategy to handle the task of evaluating WHIZ and COCOMO using DE-A's experience is to conduct a controlled experiment in which the DE-A project is undertaken two more times under the exact same conditions—except that in one case, it would be initiated using the WHIZ estimates (3,500 and 237), while in the other COCOMO's estimates (1,305 and 237) would be used. While theoretically possible, such an option is, however, infeasible in practice. A more practical alternative, is to use simulation experimentation. In addition to permitting less costly and less time-consuming experimentation, simulation makes perfectly controlled experiments possible.

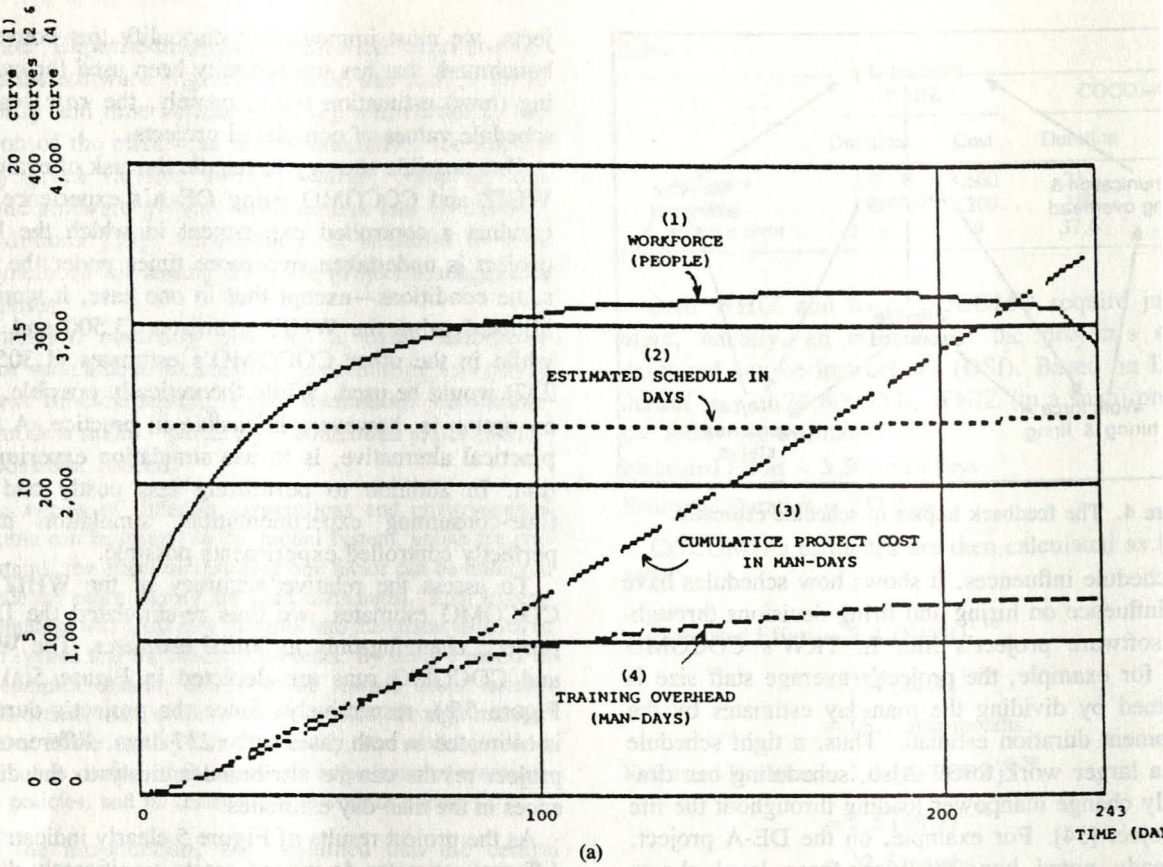
To assess the relative accuracy of the WHIZ and COCOMO estimates, we thus re-simulated the DE-A project, changing only its initial estimates. The WHIZ and COCOMO runs are depicted in Figure 5(a) and Figure 5(b), respectively. Since the project's duration is estimated in both cases to be 237 days, differences in project results can be attributed entirely to the differences in the man-day estimates.

As the project results of Figure 5 clearly indicate, the different estimates do indeed create significantly different project dynamics. The difference between the work force patterns is particularly striking. Because WHIZ generates a larger man-day estimate for the project, a larger work-force level is assembled early on, and it remains relatively stable throughout the life of the project. Contrast this with Figure 5(b), where the work-force level is significantly lower throughout most of the life cycle, and then rises steadily late in the cycle (as also happens in the base-case).

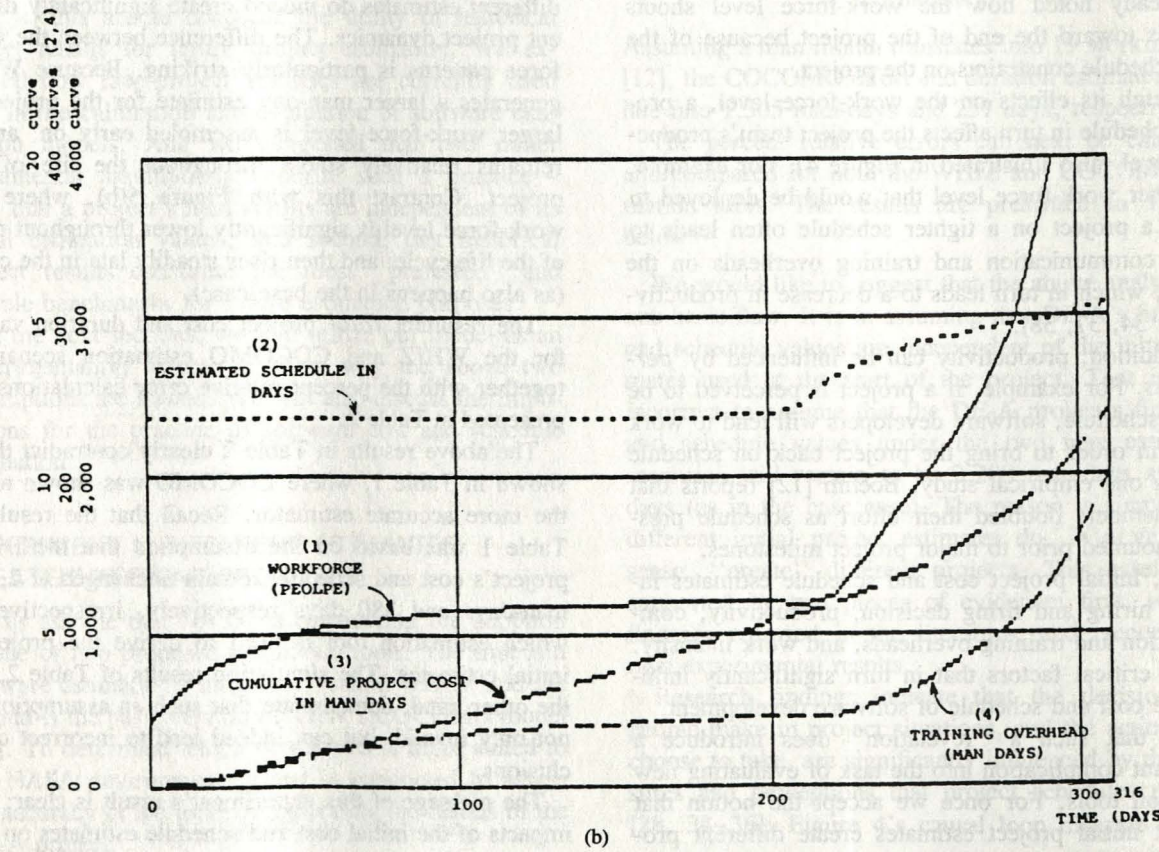
The resultant *total* project cost and duration values for the WHIZ and COCOMO estimation scenarios, together with the percent relative error calculations are presented in Table 2.

The above results in Table 2 clearly contradict those shown in Table 1, where COCOMO was shown to be the more accurate estimator. Recall that the result of Table 1 was based on the assumption that the DE-A project's cost and schedule remain unchanged at 2,200 man-days and 380 days respectively, irrespective of which estimation tool is used to derive the project's initial estimates. The simulation results of Table 2, on the other hand, demonstrate that such an assumption is not only invalid, but can indeed lead to incorrect conclusions.

The message of this experiment's result is clear: the impacts of the initial cost and schedule estimates on the



(a)



(b)

Figure 5. (a) The WHIZ simulation run; (b) the COCOMO simulation run.

J. SYSTEMS SOFTWARE
1990; 13:71–82

Table 2.

	WHIZ		COCOMO	
	Duration	Cost	Duration	Cost
Initial estimates	237	3,500	237	1,305
Final values	243	3,516	316	2,588
% Relative error	2.5	0.5	25	49.6

dynamics of a software project are both real and significant. Ignoring their influence in the ex-post evaluation of estimation tools is, thus, an approximation that cannot be justified.

5. EXPERIMENT 2: WHY FITTING TO RAW HISTORICAL VALUES MAY NOT BE "FITTING"

The objective of this second experiment is to challenge the notion that raw historical project values constitute the one (and only) preferred benchmark for fitting, calibrating, and fine-tuning estimation models. Again, as in experiment 1, we will use the DE-A project as a case example for refuting the above notion.

Recall that DE-A's size was initially underestimated to be 16,000 DSI (instead of 24,400 DSI). According to Boehm [12], the tendency to underestimate the size of a new software system is pervasive in the software industry. A major cause for undersizing is the "powerful tendency to focus on the highly visible mainline components of the software, and to underestimate or completely miss the unobtrusive components (e.g., help message processing, error processing, and moving data around)" [12].

Schedule estimation models are garbage in–garbage out devices: If poor sizing data is input in one side, poor schedule estimates come out the other side. On the DE-A project, the initial 35% underestimation of project size does indeed lead to an underestimate of the project's man-day and time requirements. As the DE-A project progressed and the level of knowledge of what the software was intended to do increased, the missed tasks were progressively discovered. This is reflected in the "Perceived Job Size in KDSI" curve in Figure 3. But, as is typically the case, newly discovered tasks do not necessarily trigger an appropriate adjustment to the project's man-day and schedule estimates. Only when the discovered "chunks" of tasks are significant in size do project members "bother" to go through the trouble of formally updating their estimates [19]. The determining factor is not the absolute size of the discovered tasks, but rather their size relative to the amount of effort perceived remaining. For example, while a 10-man-day task discovered at the beginning of a 2,000-man-day project might not trigger any adjustments in the project's estimates, it would be quite unlikely for this to happen if the 10-man-day task is discovered at the end of the development phase when only 20 man-days were remaining.

A combination of programmers' optimism and the less-than-perfect accuracy in measuring development progress early in the life cycle mask the deficit in the project's man-day requirements that result from the above practice [30, 34]. The classic result is the "90% syndrome" phenomenon, namely, where estimates of the fraction of work completed increase as originally planned until a level of 80%–90% is reached, and then they increase only very slowly until the project is actually completed. This was clearly manifested on the DE-A project as demonstrated by the "Estimated Percent Complete" curve of Figure 3.

Toward the end of the development phase, though, the discrepancy between the initial optimistic estimates and the project's true requirements becomes increasingly visible. As this happened on the DE-A project, management realized that many more tasks remain to be done and precious little time left to do them. Their response was that dramatic increase in staff size. However, as was explained earlier, adding staff members that late in the project life cycle can be a costly strategy.

We may now suspect that DE-A's cost of 2,200 man-days may *not* be a desirable benchmark, since it reflects the inefficiencies incurred in the staffing of the project, which in turn were a result of the initial undersizing of the project. Thus, if a new project comes along that happens to be exactly similar to DE-A, and if we assume that its size is properly estimated at the start, then a more effective staffing plan would be devised that avoids DE-A's last-minute staff explosion. As a result the new project *should* require less than 2,200 man-days to accomplish.

Notice, we used the word *should* and not *would*. For, if DE-A's (inflated) 2,200 man-days value is in fact adopted as the benchmark for estimating the new 24,400 DSI project, such savings may indeed *not* be realized. The reason: the self-fulfilling prophecy of Parkinson's law. On a software project, work can expand in many different forms to fill the available time. For example, work expansion could take the form of goldplating (e.g., adding features to the software product that make the job bigger and more expensive but that provide little utility to the user or maintainer when put into practice), or it could be in the form of an increase in people's slack-time activities (such as catching up on the mail, coffee breaks, etc.) [12].

Therefore, what is needed is a strategy that allows us to "wring" out those man-day excesses from the DE-A project (because of undersizing), and thus derive an

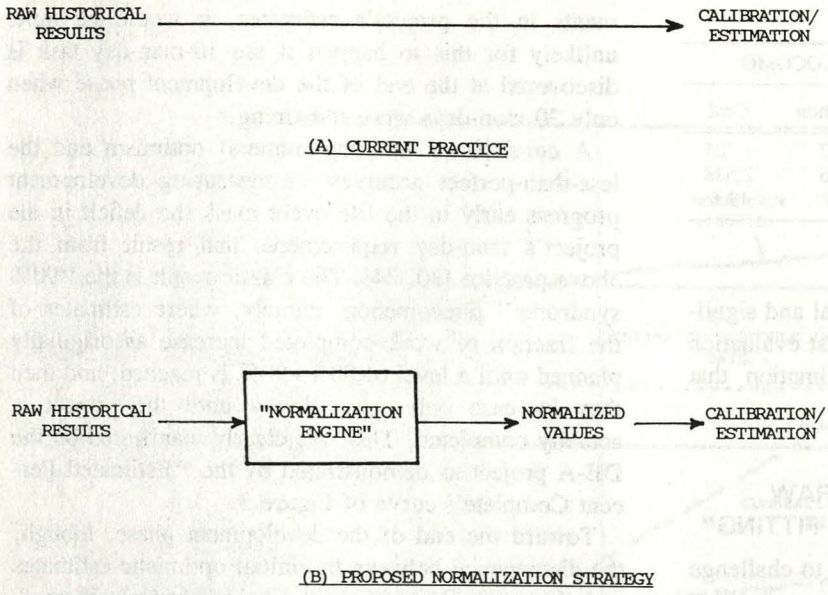


Figure 6. (a) Current practice; (b) proposed normalization strategy.

ex-post-set of *normalized* cost and schedule estimation benchmarks (Figure 6[b]). In the remainder of this section, we demonstrate how the system dynamics simulation model of software project management proposed in this article is utilized to accomplish this.

The strategy involves re-simulating the DE-A project with *no* undersizing. In order to determine the extent of the man-day excesses, not one but several simulation runs were conducted in which the initial schedule estimate is held constant at 380 days, while the man-day estimate is gradually decreased to lower values. The results of such an experiment are shown in Figure 7. The X-axis depicts the different initial man-day estimates, while the Y-axis depicts the project's final (simulated) cost in man-days.

The results indicate that using DE-A's (inflated) raw value of 2,200 is indeed wasteful. As the initial man-day estimate for the project is gradually lowered, savings are achieved, as wasteful project practices such as goldplating, unproductive slack time activities, etc. are gradually shrunk. This continues until the 1,900 man-day level is reached. Lowering the project's initial man-day estimate below this point, however, becomes counterproductive, as the project not only sheds off all its excess but also becomes in effect an underestimated project. Initial underestimation is costly (whether it is due to initial undersizing or not), as it often leads to an initial understaffing, followed by a costly staff buildup later in the lifecycle.

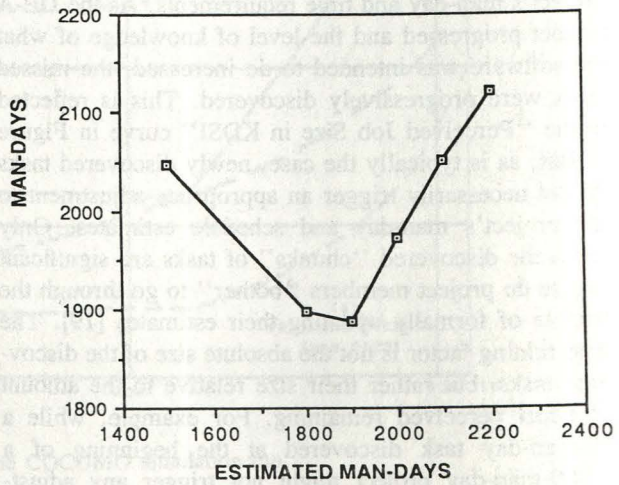
The above results, thus, suggest that the widely held notion that *raw* historical project results constitute the most "preferred" benchmark for future estimation is not only flawed but can be costly as well. In the particular case of NASA's DE-A project, a 1,900 man-day value is clearly a more "preferred" benchmark over DE-A's raw 2,200 value, as it would save NASA 234 man-days—a 10.6% saving in cost.

CONCLUSION

Three lessons can be drawn from the results of this research. First, that it is inadequate to assess the accuracy of (new) estimation tools simply on the basis of how accurately they replicate old projects. Second, we showed why raw historical project results do not necessarily constitute the most "preferred" and reliable benchmark for future estimation.

Third, the system-dynamics-based simulation approach adopted in this study proved to be a viable research vehicle for the study of software project esti-

Figure 7. Experiment 2 results.



mation. In addition to permitting less costly and less-time-consuming experimentation, simulation-type models make "perfectly" controlled experimentation possible. Furthermore, the model provided insight into the causes behind the different behavior patterns observed. In experiment 1, we showed how the model can be used as a platform to try out new estimation tools, observe their impact, and evaluate their accuracy. And in experiment 2, the model was used to wring out man-day excesses incurred as a result of initial undersizing and, hence, derive a *normalized* set of benchmarks for future estimation.

Our future research will focus on operationalizing and institutionalizing the above ideas. This will involve incorporating some extensions to the model to expand its applicability to a larger class of projects at one of NASA's software development centers. The model will then be used to normalize an extensive database of raw historical results. A long-term study will then be undertaken to compare the performance of estimators that are calibrated using the normalized database vis-à-vis estimators calibrated using the database of raw historical values.

REFERENCES

1. J. A. Farquhar, A Preliminary Inquiry into the Software Estimation Process, tech. rep, AD F12 052, Defence Documentation Center, Alexandria, Va., August 1970.
2. B. W. Boehm, Software Engineering Economics, *IEEE Trans. Software Engineering* January, 4-21 (1984).
3. R. K. B. Black, et al., BCS Software Production Data, Boeing Computer Services, Inc., Final Technical Report, RADC-TR-77-116, NTIS AD-A039852, March 1977.
4. C. E. Walston and C. P. Felix, A Method of Programming Measurement and Estimation, *IBM Systems J.*, 16, 54-73 (1977).
5. L. H. Putnam, A General Empirical Solution to the Macro Software Sizing and Estimating Problem, *IEEE Trans. Software Engineering* July, 345-361 (1978).
6. W. M. Carriere and R. Thibodeau, Development of a Logistics Software Cost Estimating Technique for Foreign Military Sales, General Research Corp., Rep. CR-3-839, June 1979.
7. F. R. Freiman and R. D. Park, PRICE Software Model —Version 3: An Overview, *IEEE PINY Workshop on Quantitative Software Models*, IEEE Cat. TH0067-9, October 1979, pp. 32-41.
8. J. J. Bailey and V. R. Basili, A Meta Model for Software Development Resource Expenditures, in *The Fifth International Conference on Software Engineering*, March 1981, pp. 107-116.
9. H. F. Dircks, SOFCOST: Grumman's Software Cost Estimating Model, *IEEE NAECON 1981* May 1981.
10. R. C. Tausworthe, Deep Space Network Software Cost Estimation Model, Jet Propulsion Lab, Pasadena, Calif., 1981.
11. R. W. Jensen, An Improved Macrolevel Software Development Resources Estimation Model, *Fifth ISPA Conference*, pp. 88-92, April 1983.
12. B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
13. R. Thibodeau, An Evaluation of Software Cost Estimating Models, General Research Corp., Rep. T10-2670, April 1981.
14. Mohanty, S. N. "Software Cost Estimation: Present and Future." *Software Practice and Experience,* Vol. 11 (1981), 103-121.
15. Barbacci, M. R., Habermann, A. N., and Shaw, M. "The Software Engineering Institute: Bridging Practice and Potential." *IEEE Software,* November 1985, 4-21.
16. M. V. Zelkowitz, et al. Software Engineering Practices in the US and Japan, *Computer* June, 57-66 (1984).
17. S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*, Benjamin/Cummings, Menlo Park, Calif., 1986.
18. C. F. Kemerer, An Empirical Validation of Software Cost Estimation Models, *Commun. ACM* 30 (5), May 1987.
19. T. K. Abdel-Hamid, The Dynamics of Software Development Project Management: An Integrative System Dynamics Perspective, unpublished Ph.D. dissertation, Sloan School of Management, MIT, January 1984.
20. T. K. Abdel-Hamid and S. E. Madnick, Modeling the Dynamics of Software Project Management, *Commun. ACM* December (1988).
21. T. K. Abdel-Hamid and S. E. Madnick, *Software Development Dynamics: An Integrated Approach*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
22. Cougar, J. D. and Zawacki, R. W. *Motivating and Managing Computer Personnel.* New York, N.Y.: John Wiley & Sons, Inc., 1980.
23. H. S. Bott, The Personnel Crunch, in *Perspectives on Information Management* (J. B. Rochester, ed.), Wiley, New York, 1982.
24. Winrow, Acquiring Entry-Level Programmers, in *Computer Programming Management* (J. Hannan, ed.), Auerbach Publishers, Pennsauken, New Jersey, 1982.
25. C. L. McGowan and R. C. McHenry, Software Management, in *Research Directions in Software Technology* (P. Wegner, ed.), MIT Press, Cambridge, Mass., 1980.
26. I. D. Steiner, *Group Process and Productivity*, Academic Press, New York, 1972.
27. J. W. Forrester, *Industrial Dynamics*, The MIT Press, Cambridge, Mass., 1961.
28. H. D. Mills, *Software Productivity*, Little, Brown & Co., Canada, 1983.
29. T. DeMarco, *Controlling Software Projects*, Yourdon Press, New York, 1982.
30. R. L. Baber, *Software Reflected*, North Holland, New York, 1982.
31. T. K. Abdel-Hamid, The Dynamics of Software Project Staffing: A System Dynamics Based Simulation Ap-

proach, *IEEE Trans. Software Engineering* February (1989).
32. Myers, W. "The Need for Software Engineering," *Computer*, February 1978.
33. R. L. Glass, *Modern Programming Practices: A Report from Industry*, Englewood Cliffs, New Jersey, 1982.
34. F. P. Brooks, *The Mythical Man Month*, Addison-Wesley, Reading, Mass., 1978.
35. R. L. Ibrahim, Software Development Information System, *J. Syst. Management* Dec., 34–39 (1978).
36. A. Radice, Productivity Measures in Software, In *The Economics of Information Processing Volume 2: Operations, Programming, and Software Models* (R. Goldberg and H. Lorin, eds.), John Wiley, New York, 1982.
37. R. F. Scott and B. D. Simmons, Programmer Productivity and the Delphi Technique, *Datamation* May, 71–73 (1974).
38. M. L. Shooman, *Software Engineering – Design, Reliability and Management*, McGraw-Hill, New York, 1983.