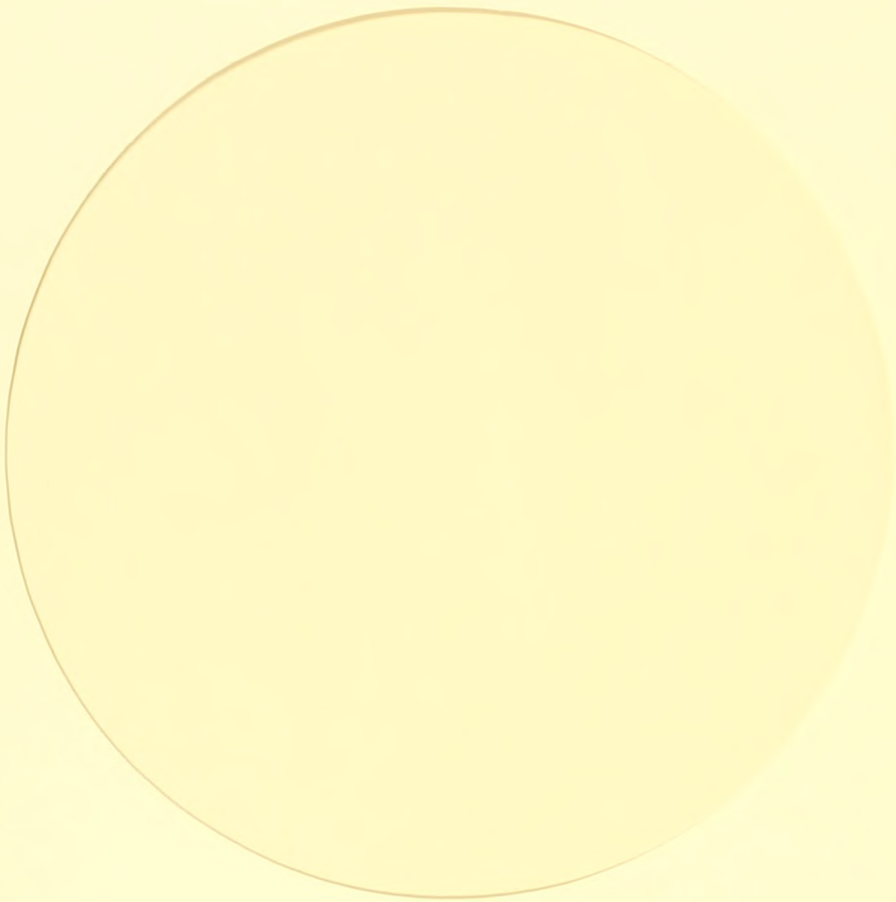
MODELING THE DYNAMICS OF SOFTWARE PROJECT MANAGEMENT

Tarek K. Abdel-Hamid
Stuart E. Madnick

September 1987

CISR WP No. 163
Sloan WP No. 1935-87

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



MODELING THE DYNAMICS OF SOFTWARE PROJECT MANAGEMENT

Tarek K. Abdel-Hamid
Stuart E. Madnick

September 1987

CISR WP No. 163
Sloan WP No. 1935-87

Submitted for publication to the *Communications of the ACM*

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

MODELING THE DYNAMICS OF SOFTWARE PROJECT MANAGEMENT

Summary

The development of software has been marked by the problems of cost overruns, late deliveries, poor reliability, and users' dissatisfaction which continue to persist in spite of the significant advances that have been made in the software engineering field to tackle the technical hurdles of software production.

The objective of this paper is to enhance our understanding of, and gain insight into the general process by which software development is managed by integrating our knowledge of its multiple activities into an integrated continuous view of the software development process.

The model is currently being used as an experimentation vehicle to study/predict the dynamic implications of an array of managerial policies and procedures pertaining to the management of software projects.

Introduction

The impressive improvements that are continuously being made in the cost-effectiveness of computer hardware are causing an enormous expansion in the number of applications for which computing is becoming a feasible and economical solution. This in turn, is placing greater and greater demands for the development and operation of computer software systems. A conservative estimate indicates a "tenfold increase in the demand for software each decade, or a hundred fold increase between 1965 and 1985" (Musa, 1985).

This growth in the demand for software has not, however, been painless. The record shows that the development of software has been marked by cost overruns, late deliveries, poor reliability, and users' dissatisfaction [(Buckley and Poston, 1984), (Ramamoorthy et al., 1984), and (Newport, 1986)].

In an effort to bring discipline to the development of software systems, attempts have been made since the early 1970s to apply the rigors of science and engineering to the software production process. On the technology side, significant progress has been made over the last decade, leading to the

development of a large number of methodologies (e.g., structured programming, structured design, formal verification, language design for more reliable coding, diagnostic compilers, and so forth) that address many of the technical problems experienced in software development.

A comparable evolution on the managerial front has not occured, however [(Zmud, 1980) and (Beck and Perkins, 1983)].

> Software engineering project management (SEPM) has not enjoyed the same progress (as the technology of software development). While it might be argued that SEPM has been defined, it is far from a recognized discipline ... The major issues and problems of SEPM have not been agreed on by the computing community as a whole, and consequently, priorities for addressing them have not been widely established. Furthermore, research in this area has been scant (Thayer et al., 1981).

This position is further substantiated by a survey, reported in the same paper, which revealed that only a handful of U.S. universities offer courses in the area of software project management.

This "deficiency" in the field's research repertoire is being blamed by a growing number of researchers and practitioners for the persistence of our difficulties in producing software that is on time, within budget, and that meets user requirements [(Pooch and Gehring, 1980), (Thomsett, 1980), and (Weinberg, 1982)]. A chief concern expressed is that, as of yet, we still lack a fundamental understanding of the software development process, and that without such an understanding the possibility or likelihood of any significant gains on the managerial front is questionable [(Basili, 1982) and (McKeen, 1983)].

This paper reports the results of an ongoing research effort designed to address the above concerns. Specifically, it is our goal to develop a comprehensive mathematical model of the software development process and to use such a model as a laboratory vehicle to study, gain insight into, and

make predictions about the software project management process.

In the remaining parts of this paper we present and discuss the integrative dynamic model of software project management that has been developed. We will provide an overview of both the model's structure and its behavior. We begin our presentation, however, by first presenting the arguments for the utility of such a formal dynamic modeling approach in the study of software project management.

The High Complexity of the Software Project Management Process

Project management often is based simplistically on a "mental picture" captured by the single-loop model shown in Figure 1 (Roberts, 1981). The model portrays how project work is accomplished through the utilization of (1) project resources (manpower, facilities, equipment). As (2) work is accomplished on the project, it is reported (3) through some project control system. Such reports cumulate and are processed to create the (4) project's forecast completion time by adding to the current date the indicated time remaining on the job. Assessing the job's remaining time involves figuring out the magnitude of the effort (e.g., in man-days) believed by management to be remaining to complete the project, the level of manpower working on the project, and the perceived productivity of the project team. The feedback loop is completed (closed) as the difference, if any, between the (5) scheduled completion date and the (4) forecast completion date causes adjustments (6) in the magnitude or allocation of the project's resources.

What is attractive about the above model is that it is both reasonable and simple. It is, therefore, a mental tool that is not too difficult to wield. But is it an adequate model for the dynamics of software project management?

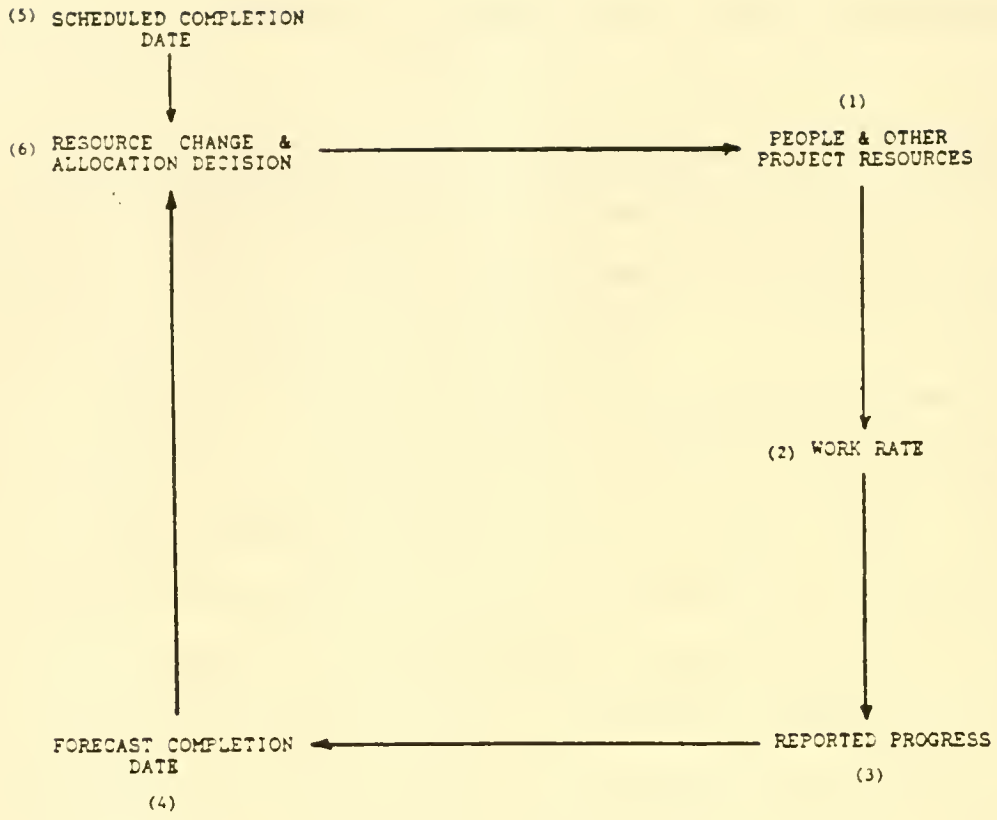


FIGURE (1)

A MODEL OF SOFTWARE PROJECT MANAGEMENT

The software project management system is a far more complex conglomerate of interdependant variables that are interrelated in various nonlinear fashions. By excluding some vital aspects of the real software project environment, the above model could seriously misguide the unsuspecting software manager. To see how, let us consider some of the typical decisions pondered in a software project environment.

Adding more people to a late project: The mental picture of Figure 1 suggests a direct relationship between adding people resources and increasing the rate of work on the project, i.e., the higher the level of project resources the higher the work rate. Subscribing to this simplistic view of the world has led many a software manager into serious trouble (Brooks, 1978).

For example, one vital aspect of software project dynamics that no software manager can afford to ignore is captured by the feedback loop of Figure 2. It portrays some of the dynamic forces that create the phenomenon known as "Brooks' Law", i.e., that adding more people to a late software project makes it later (Brooks, 1978). As the figure indicates, adding more people often leads to higher communication and training overheads on the project, which can in turn dilute the project team's productivity. Lower productivity translates into lower progress rates, which would delay the late project even further. This in turn could trigger an additional round of workforce additions and another pass around this "vicious cycle."

In Figure 3a we, therefore, amend Figure 1 by incorporating the vital link between the workforce level (and the associated communication and training overheads) and productivity.

Adjusting the Schedule of a Late Project: Another part of the real system ignored by Figure 1 is the human element in project actions and decisions. The attitudes and motivations of software developers and their managers,

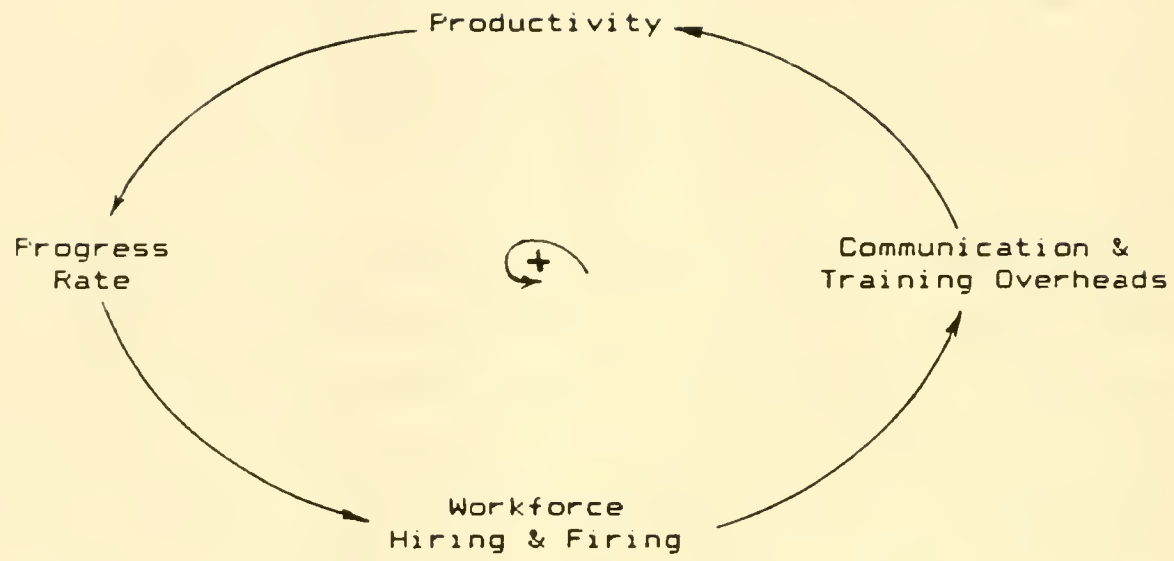
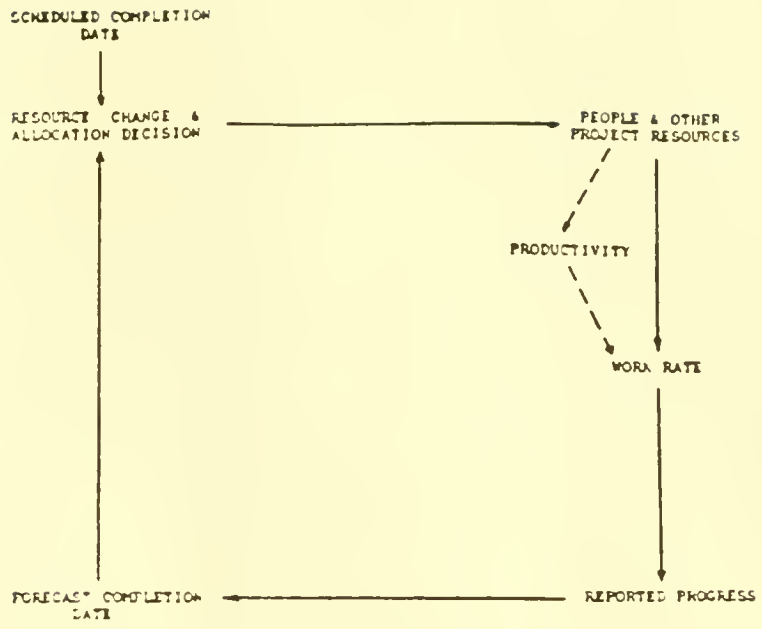
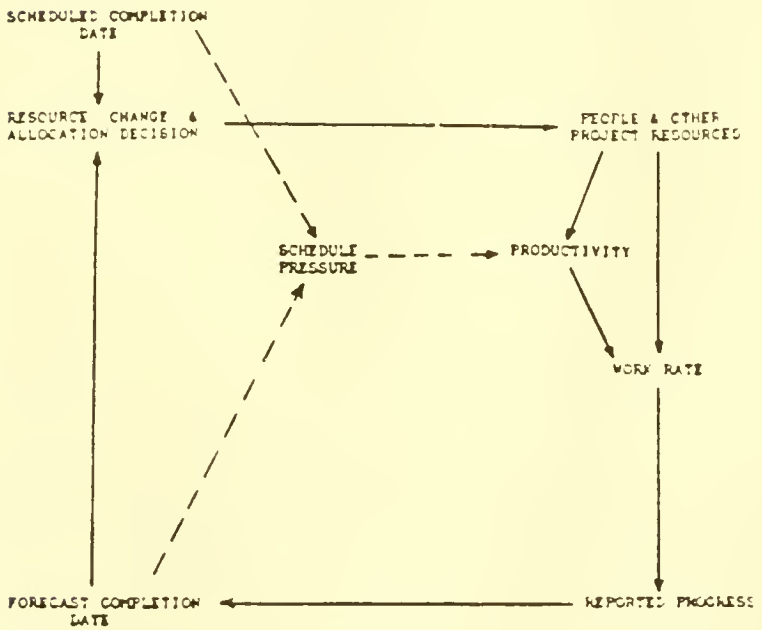


Figure (2)

EXAMPLE FEEDBACK LOOP



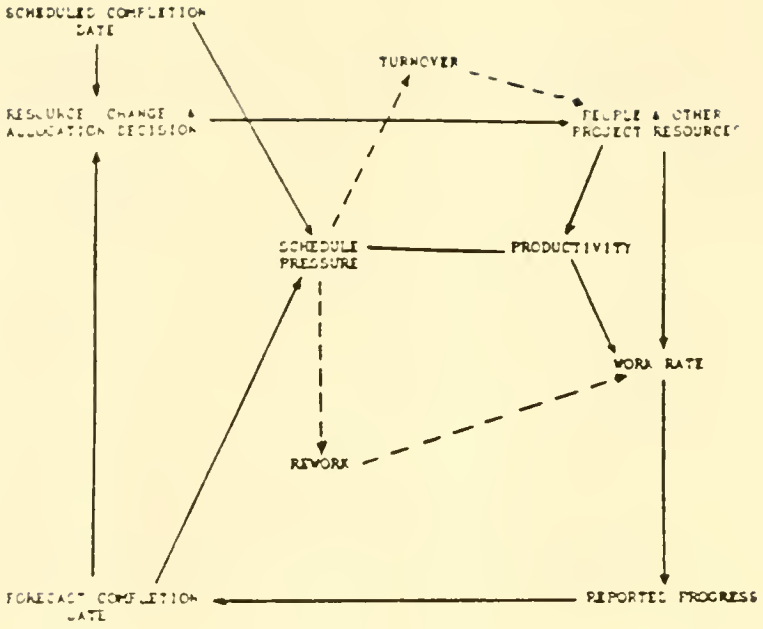
(a)



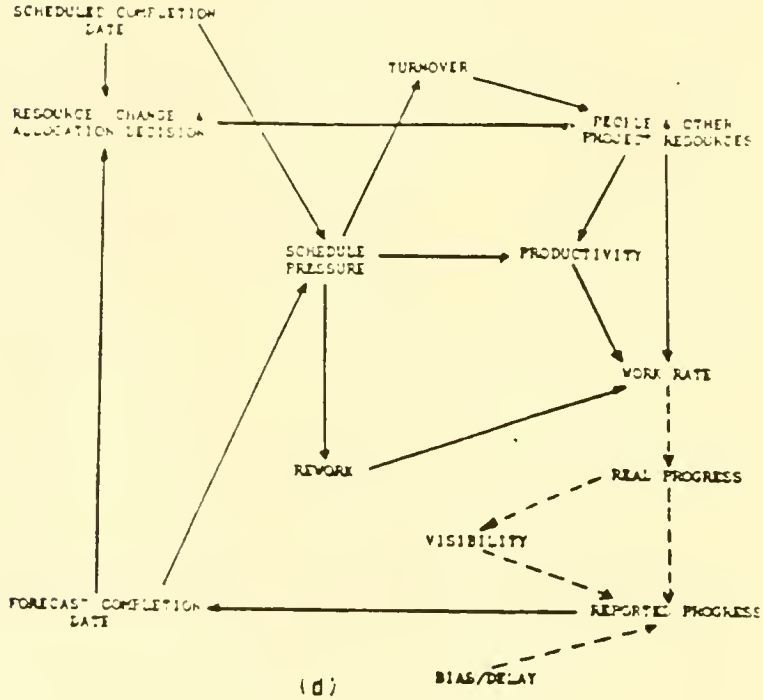
(b)

Figure (3)

REFINEMENTS TO THE MODEL OF SOFTWARE PROJECT MANAGEMENT



(c)



(d)

Figure (3) (continued)

their knowledge of the schedules, and current estimates in the project, all affect the real progress that is achieved, as well as the progress and problems that are reported upward in the organization.

For example, when faced with schedule pressures that arise as a project falls behind its schedule, software developers typically respond by putting in longer hours and by concentrating more on the essential tasks of the job (Ibrahim, 1978). In one experiment, Boehm (1981) found that the number of man-hours devoted to project work increased by as much as 100%. Most of this gain was achieved by reallocating people's slack time by spending less time on off-project activities such as personal business, coffee breaks, non-project commmunication.

This link between schedule pressure and productivity is captured in Figure 3b. Does this mean, then, that software managers need not worry about adjusting the schedule of a late project and should rely, instead, on simply maintaining the pressure on their project teams?

The impact of schedule pressures on software development is not limited to the above relatively direct role. Schedule pressures can also play less visible roles. For example, as Figure 3c suggests, schedule pressures can increase the error rate of the project team and thus the amount of rework on the project [(Radice, 1982) and (Mills, 1983)].

> People under time pressure don't work better, they just work faster ... In the struggle to deliver any software at all, the first casuality has been consideration of the quality of the software delivered (DeMarco, 1982).

The rework necessary to correct such software errors obviously diverts the project team's effort from making progress on new project tasks, and thus can have a significant negative impact on the project's progress rate.

Also, consider the impact of schedule pressure on the workforce turnover rate (Figure 3c). There is evidence to suggest that workforce turnover

increases when unreasonably tight scheduling situations persist in an organization (Freedland, 1987). This can be quite costly, since a higher turnover rate translates into lower productivity on the project.

Finally, How Really Late is a Late Software Project? Before a software manager can succeed in rescuing a lagging software project and bringing it back on track by adding people, adjusting the schedule, etc., it is imperative that the assessment of the magnitude of the delay be on target. But software is basically an intangible product during most of the development process. This lack of visibility can produce a significant difference between the real achievement and the perceived progress on the job. To the extent that the perceived progress rate differs from the real progress rate, an error in perceived cumulative progress will gradually accumulate (Figure 3d). This undoubtedly poses yet another complication that is too real for the software project manager to exclude from any model or analysis of the process.

A Need for an Integrative Perspective of Software Development

The above discussion illustrates that there are a large number of variables, both tangible and intangible, that impact the software development process. Furthermore, these variables are not independent, but are related to one another in complex fashions. Perhaps most importantly, understanding the behavior of such systems is complex far beyond the capacity of human intuition (Roberts, 1981).

A major deficiency in much of the research to date on software project management has been its inability to integrate our knowledge of the micro components of the software development process such as scheduling, progress measurement, and staffing to derive implications about the behavior of the total socio-technical system in which the micro components are embedded

(Thayer, 1979). In the words of Jensen and Tonies (1979): "There is much attention on individual phases and functions of the software development sequence, but little on the whole lifecycle as an integral, continuous process."

The model presented in this paper provides such an integrative perspective. It integrates the multiple functions of the software development process, including both the management-type functions (e.g., planning, control, staffing) as well as the software production-type activities (e.g., design, coding, reviewing, testing).

A second unique feature of our modeling approach is the use of the feedback principles of System Dynamics to structure and clarify the complex web of dynamically interacting variables involved in the development and management of software projects. Feedback is the process in which an action taken by a person or thing will eventually affect that person or thing. Examples of such feedback systems in the software project environment have already been demonstrated in the above discussion and are evident in Figures 1 through 3.

The significance and applicabilty of the feedback systems concept to managerial systems has been substantiated by a large number of studies (Roberts, 1981). For example, Weick (1979) observes that,

> The cause-effect relationships that exist in organizations are dense and often circular. Sometimes these causal circuits cancel the influences of one variable on another, and sometimes they amplify the effects of one variable on another. It is the network of causal relationships that impose many of the controls in organizations and that stabilize or disrupt the organization. It is the patterns of these causal links that account for much of what happens in organizations. Though not directly visible, these causal patterns account for more of what happens in organizations than do some of the more visible elements such as machinery, timeclocks, ...

The third distinctive aspect of our modeling approach is the utilization of the computer simulation tools of System Dynamics to handle the high

complexity of the resulting integrative feedback model. The behavior of systems of interconnected feedback loops often confounds common intuition and analysis, even though the dynamic implications of isolated loops may be reasonably obvious. The feedback structures of real problems are often so complex that the behavior they generate over time can usually be traced only by simulation (Richardson and Pugh, 1981).

Before describing the model and experiments performed, there are several points that are important to clarify. First, due to its length and complexity only a portion of the entire model can be presented and explained in this paper. For more details the reader is referred to [(Abdel-Hamid, 1984) and (Abdel-Hamid and Madnick, 1987a)]. Second, the focus of this research is on the dynamics of software projects; that is, aspects that change during the life of the project, such as workforce level and productivity, rather than aspects that are decided once and then remain constant throughout the project, such as choice of programming language.

Third, it is necessary to have a perspective about what the model is and is not intended to accomplish. This is particularly relevant because this research is primarily intended to provide understanding of the dynamic behavior of a project (e.g., how variables like workforce-level and productivity change over time and why) rather than to provide point-predictions (e.g., of the number of errors generated).

Fourth, although the model was developed based on field studies and a careful literature review, a model is, by definition, a simplification. Thus a model's value ultimately depends upon its ability to help us understand an otherwise overly complex situation. Specific benefits are that unlike a mental model, a System Dynamics simulation model can reliably trace through time the implications of a messy maze of assumptions and interactions,

without stumbling over phraseology, emotional bias, or gaps in intuition.

Model Structure

Our integrative dynamic model of software project management was developed on the basis of a battery of 27 field interviews of software project managers in five software producing organizations, supplemented by an extensive database of empirical findings from the literature. Figure 4 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 interrelationships among the four subsystems.

The Human Resource Management Subsystem:

The Human Resource Management Subsystem captures the hiring, training, assimilation, and transfer of the human resource, as shown in Figure 5.

The schematic conventions used in Figure 5 are the standard conventions used in System Dynamics models. From a System Dynamics perspective all systems can be represented in terms of "level," "rate," and "auxiliary" variables.

A level is an accumulation, or an integration, over time of flows or changes that come into and go out of the level. The term "level" is intended to invoke the image of the level of a liquid accumulating in a container. The flows increasing and decreasing a level are called rates. Thus, "NEWLY HIRED WORKFORCE" is a level of people that is increased by the "HIRING RATE" and decreased by the "WORKFORCE ASSIMILATION RATE."

Rates and levels are represented as stylized valves and tubs, as shown below, further emphasizing the analogy between accumulation processes and the flow of a liquid.

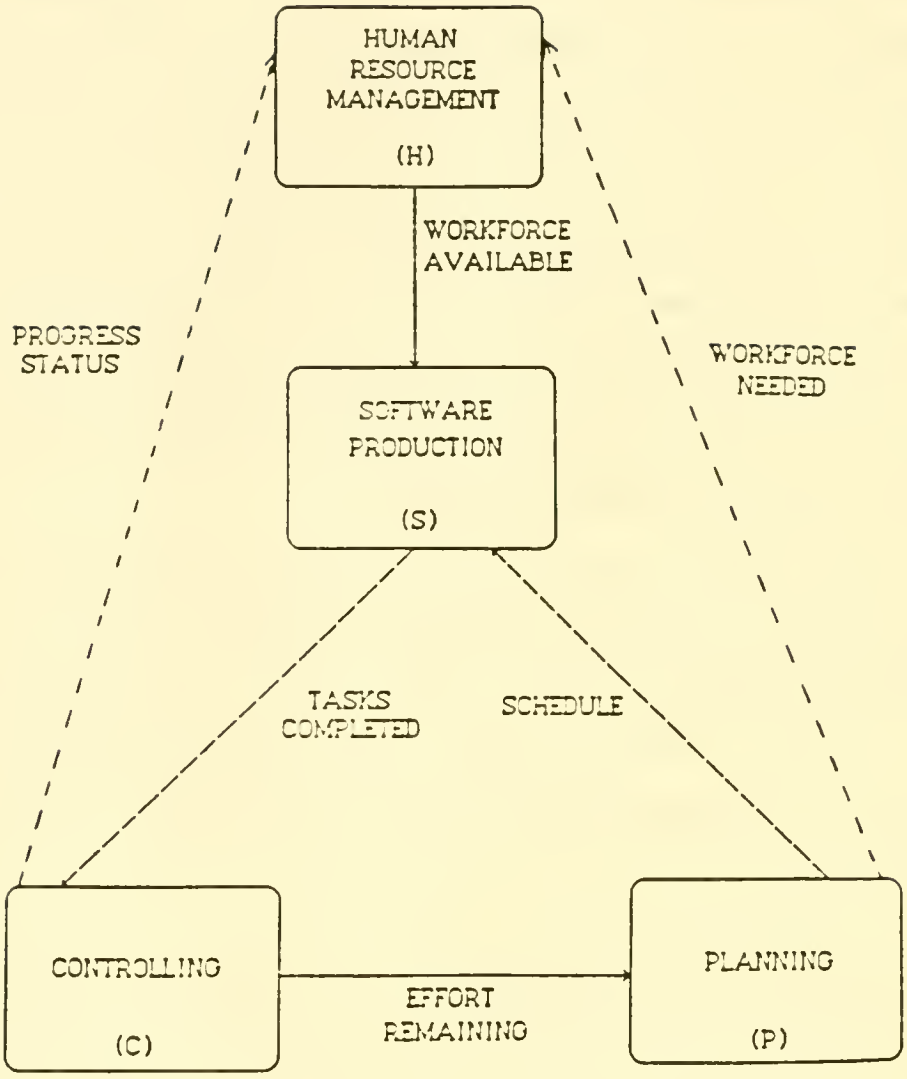


Figure (4)

Four Subsystems of Model

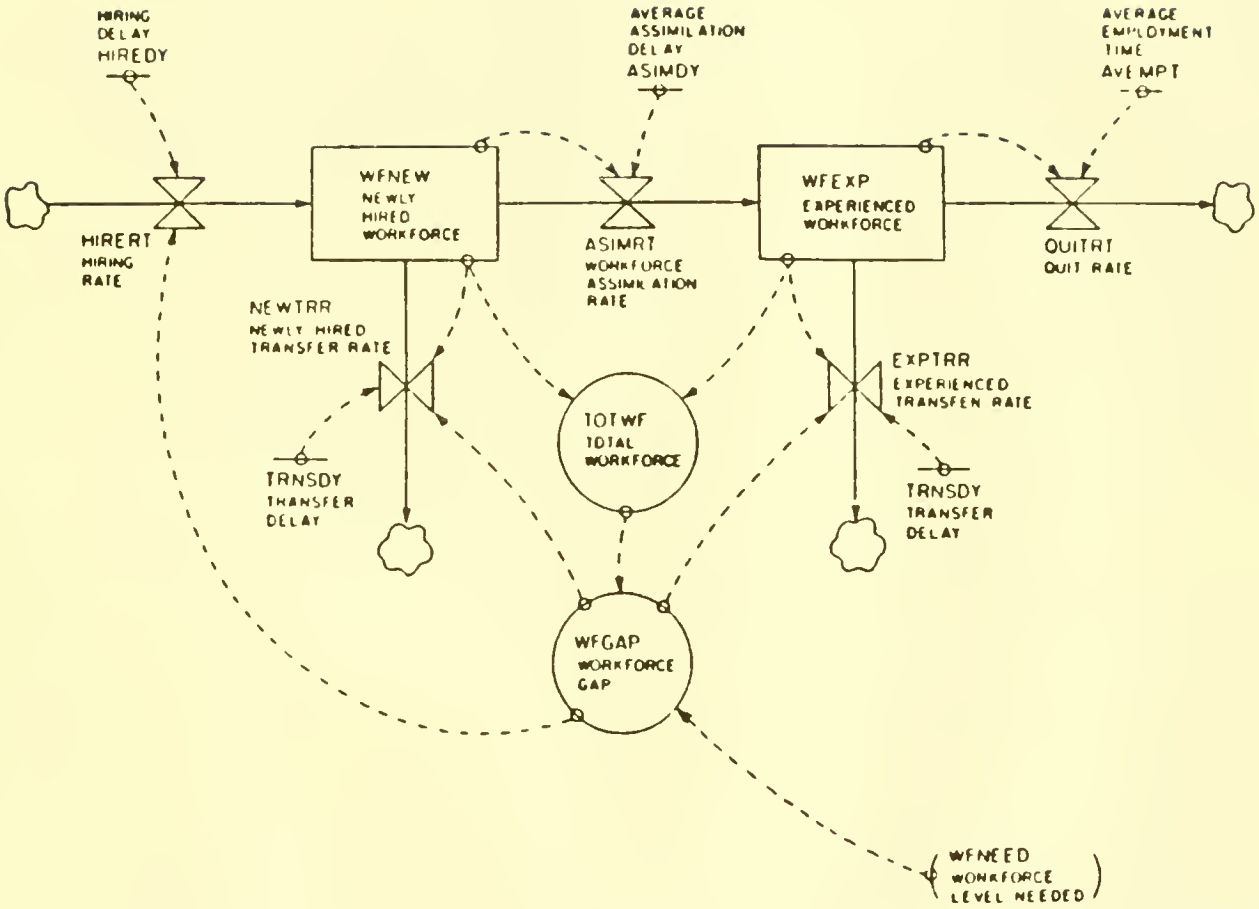
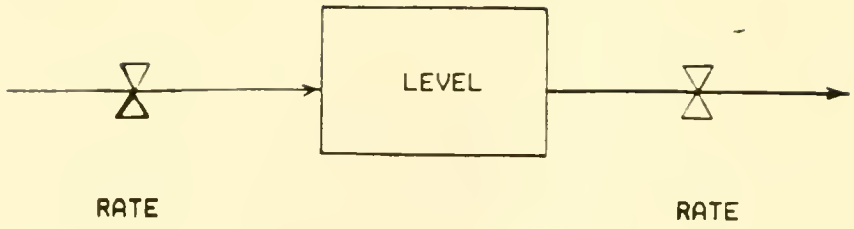

HIRING DELAY
HIREDY
AVERAGE ASSIMILATION DELAY
ASIMDY
AVERAGE EMPLOYMENT TIME
AVEMPT
WFNEW
NEWLY HIRED WORKFORCE
WFEXP
EXPERIENCED WORKFORCE
HIRERT
HIRING RATE
ASIMRT
WORKFORCE ASSIMILATION RATE
QUITRT
QUIT RATE
NEWTRR
NEWLY HIRED TRANSFER RATE
EXPTRR
EXPERIENCED TRANSFER RATE
TOTWF
TOTAL WORKFORCE
TRNSDY
TRANSFER DELAY
TRNSDY
TRANSFER DELAY
WFGAP
WORKFORCE GAP
WFNEED
WORKFORCE LEVEL NEEDED

Figure 5



The flows that are controlled by the rates are usually diagrammed differently, depending on the type of quantity involved. We will use the two types of arrow designators shown below:

INFORMATION FLOWS ----------)

OTHER FLOWS ——————)

(e.g., People)

All tangible variables are either levels or rates, i.e., they are either accumulations of previous flows or are presently flowing. Auxiliary variables, on the other hand, are information-type variables in the system, and capture things like concepts (e.g., the concept of a "WORKFORCE GAP") and policies (e.g., the policy for allocating "DAILY MANPOWER FOR TRAINIING"). Auxiliary variables are represented by a circular symbol.

Finally, variables that are defined in other sectors of the model are represented by enclosing the variable name in parentheses as shown below.

(VARIABLE FROM ANOTHER SECTOR)

Notice that the project's total workforce in Figure 5 is comprised of two workforce levels, namely, "NEWLY HIRED WORKFORCE" and "EXPERIENCED WORKFORCE." Disaggregating the workforce into these two categories of employees is necessary for two reasons. First, newly added team members are less productive (on the average) than the "old timers" (Cougar and Zawacki, 1980). Secondly, it allows us to capture the training processes involved in assimilating the new members into the project team.

On deciding upon the total workforce level desired, project managers consider a number of factors. One important factor, of course, is the current scheduled completion date of the project. As part of the planning subsystem (to be discussed later), management determines the workforce level that it believes would be necessary to complete the project within the scheduled completion time. In addition to this, however, consideration is also given to the stability of the workforce. Thus, before adding new project members, management typically contemplates the duration for which the new members will be needed. Different organizations weigh this factor to various extents. In general, the relative weighing between the desire for workforce stability on the one hand and the desire to complete the project on time, on the other, changes with the stage of project completion. For example, toward the end of the project there could be considerable reluctance to bring in new people, even though the time and effort perceived remaining might imply that more

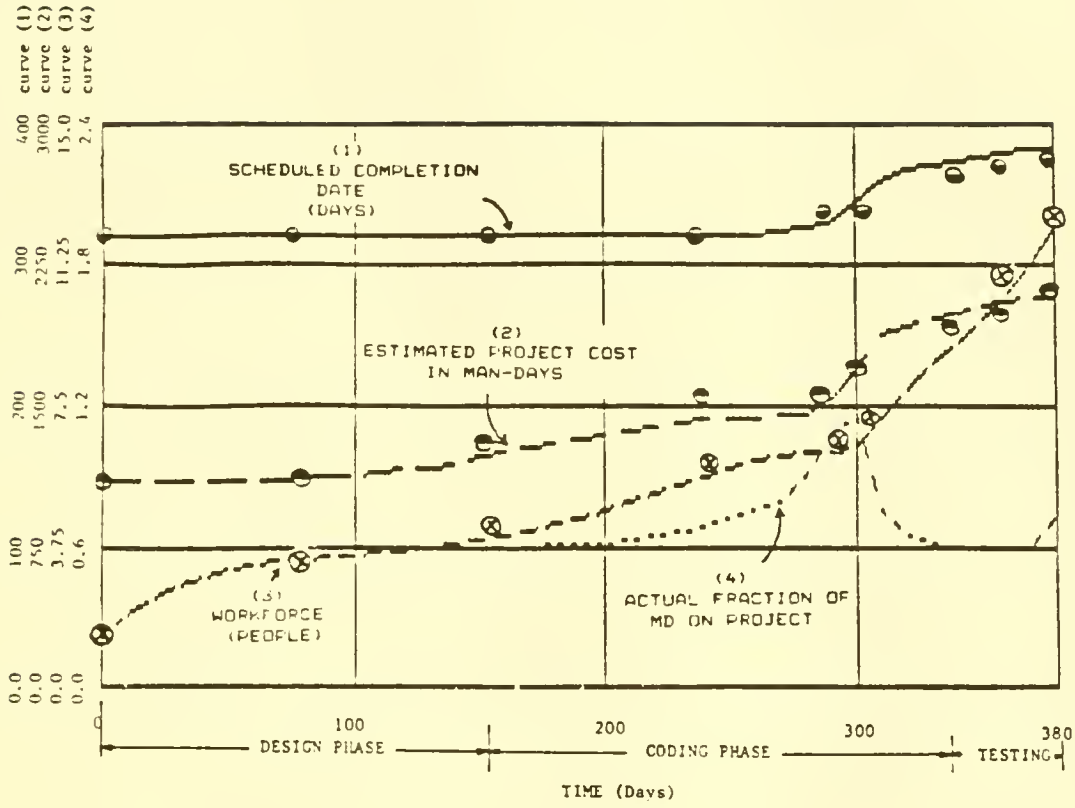
people are needed. This reluctance would arise from the realization that there just wouldn't be enough time to acquaint the new people with the mechanics of the project, integrate them into the project team, and train them in the necessary technical areas.

Figure 6 demonstrates the kinds of dynamic behaviors reproduced by the model. It depicts the model's output that resulted from simulating the behavior of one of NASA's software projects, the DE-A software project (NASA, 1983). (See the Appendix for more details on the DE-A project.) The model's results conformed quite accurately to the project's actual behavior (represented by the O points in the figure), as is documented in detail in (Abdel-Hamid, 1984).

Notice that the workforce pattern here differs from the "typical" workforce pattern as discussed in the literature, i.e., the concave type curve that rises, peaks, and then drops back to lower levels as the project proceeds towards the system testing phase (Boehm, 1981). The reason why the workforce level here shoots upwards towards the end of the project has to do with NASA's tight scheduling constraints. Because NASA's launch of the DE-A satellite was tied to the completion of the DE-A software, serious schedule slippages were not tolerated. Specifically, all software was required to be accepted and frozen 90 days before launch. As this date was approached, pressures developed that overrode the workforce stability considerations. That is, project management became increasingly willing to "pay any price" necessary to avoid overshooting the 90-day-before-launch date. This translated, as the figure indicates, into a management that was increasingly willing to add more people.

The Software Production Subsystem

The four primary software production activities are: development, quality assurance, rework, and testing. The development activity comprises both the



◒ DE-A's actual "SCHEDULED COMPLETION DATE " (Days)
◓ DE-A's actual "ESTIMATED PROJECT COST IN MAN-DAYS"
⊗ DE-A's actual "WORKFORCE" (Full-Time Equivalent People)

FIGURE 6

OUTPUT FOR MODEL OF NASA DE-A SOFTWARE PROJECT

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

The Software Production Subsystem is too complex to fully explain in the limited space of this paper, but rather than provide just a high level overview of the total subsystem we will, instead, discuss in some detail the structure of one of the subsystem's important components, namely, the structure that captures the dynamics of software productivity.

The model's software productivity structure is depicted in Figure 7. It is based, in part, on a model of group productivity proposed by the psychologist Ivan Steiner (1972). Steiner's model can simply be stated as follows:

Actual Productivity = Potential Productivity - Losses Due to Faulty Process

where losses due to faulty process basically refer to a group's communication and motivation losses. Paraphrasing Steiner (1972):

> Potential productivity is defined as the maximum level of productivity that can occur when an individual or group ... makes the best possible use of its resources (that is, if there is no loss of productivity due to faulty process) ... Potential productivity can be inferred from a thorough analysis of task demands and available resources, for it depends only upon these two types of variables. Actual productivity, what the individual or group does in fact accomplish, rarely equals potential productivity. Individuals and groups usually fail to make the best possible use of their available resources. Problems of coordination and/or motivation are responsible for inadequacies in process, and for consequent losses in productivity.

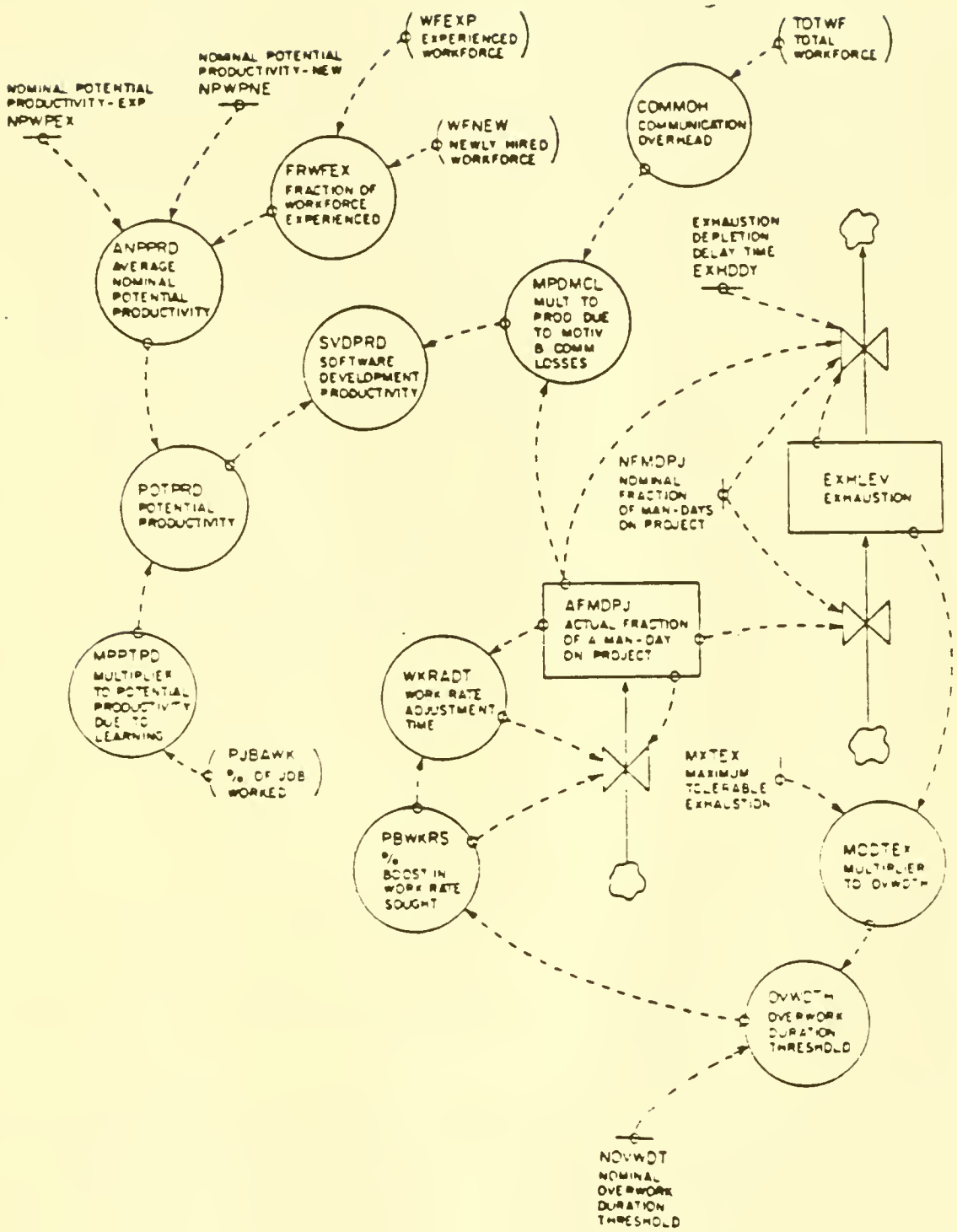


Figure 7

SOFTWARE PRODUCTIVITY SECTOR

Thus, according to Steiner, potential productivity is a function of two sets of factors, the nature of the task and the group's resources. The effects of various factors belonging to these two sets of determinants on the productivity of software development have been widely investigated in the literature. These include factors such as product complexity and database size (examples of task-type variables) and personnel capabilities and the availability of software development tools (examples of resource-type variables).

Notice that while most of the above factors would tend to vary from organization to organization (e.g., availability of software tools, personnel capability, and computer-hardware characteristics) and from project to project within a single organization (e.g., product complexity, database size, and programming language) they, however, would tend to remain constant throughout the development life of of any one particular project. This observation is quite significant for this discussion. It means that in studying the dynamic patterns of software productivity during the lifecycle of a particular software project, which is our concern here, the above variables would tend to remain constant and, therefore, would not play a dynamic role. Thus, in representing the dynamics of software productivity throughout the life of a software project such factors could be captured by a single invariant paramter. In our model this is termed the "NOMINAL POTENTIAL PRODUCTIVITY" parameter.

Not all determinants of potential productivity are, however, of such a non-dynamic nature. Two variables identified in the literature that do play a dynamic role are the workforce experience level (Chrysler, 1978) and increases in project familiarity due to learning [(Shell, 1972) and (Weinberg, 1982)]. These dynamic variables are captured in Figure 7.

Due to the losses incurred in communication and motivation overheads, actual productivity rarely equals potential productivity. Communication overhead is the drop in the productivity of the average team member caused by the losses incurred in communicating with others on the project. The nature of the relationship between communication overhead and team size has been investigated by several authors. For example, it has been suggested that communication overhead increases in proportion to n^2, where n is the size of the team [(Brooks, 1978), (Zelkowitz, 1978), and (Shooman, 1983)].

To understand the effects of motivation losses on productivity we need to make the same distinction we made above between factors that remain constant during the life of any one particular project and those that tend to change throughout the life of a project. Many of the motivational factors discussed in the literature (e.g., possibility for growth and advancement, responsibility, and salary) are factors that tend to characterize the overall organizational setting and climate. In our formulation, such non-dynamic factors would be implicitly incorporated within the definition of the potential productivity parameter.

On the other hand, goals and schedules play a dynamic motivational role throughout the life of a software project. Boehm (1981) suggested that the motivational role of schedule pressures and project deadlines is specifically to expand or contract the project members' "slack time." Slack time is the fraction of project time lost on off-project activities, e.g., coffee-breaks, personal business, and non-project communication.

The motivation mechanism in the model captures such dynamic motivational impacts of schedule pressures on "slack time." In the absence of schedule pressures the fraction of daily hours allocated (on the average) by a full-time team member to project-related work is captured by the parameter "NOMINAL FRACTION OF A MAN-DAY ON PROJECT." Several studies indicate that the

value of this parameter lies within the 50-70% range [e.g., see (Brooks, 1978), (Pooch and Gehring, 1980), and (Boehm, 1981)]. For example, a 60% value implies that an employee allocates, on the average, 0.6 X 8 = 4.8 hours to the project (assuming an 8-hour day). Under such conditions, the loss amounts to a 40% cut in potential productivity.

The loss in productivity due to motivational factors does not, of course, remain constant throughout the life of the project. The motivational effects of schedule pressures can push the "ACTUAL FRACTION OF A MAN-DAY ON PROJECT" to both a higher value (under positive schedule pressure) as well as a lower value (under negative schedule pressure).

When positive schedule pressures build up in a late project, software developers tend to work harder by compressing their slack time in an attempt to compensate for the perceived deficit, and to bring the project back on schedule [(Boehm, 1981), (DePree, 1984), and (Ibrahim, 1978)].

But what if such a situation persists? Would workers be willing to work harder indefinitely? The answer, based on our own field study results, was overwhelmingly no (Abdel-Hamid, 1984). Our findings indicate that there is a threshold for how long employees would be willing to work at an "above-normal" rate. In other words, workers need their slack time and they typically would not tolerate a prolonged deprivation of such "breathers." Compressed slack time, therefore, exhausts them (psychologically more so than physically) in the sense that it cuts into their tolerance level for continued over-working. [A significant portion of the productivity structure in the model is devoted to handling these intangible dynamic forces. The interested reader should refer to (Abdel-Hamid, 1984) or (Abdel-Hamid and Madnick, 1987a).]

The dynamic behavior of the "ACTUAL FRACTION OF A MAN-DAY ON PROJECT" for the DE-A project is depicted in Figure 6. Notice the "spike" that occurs as the end-of-development milestone is approached. To understand why this happens we need first to observe (in Figure 6) that when project DE-A started, management had underestimated its true size by 45%. As the project developed, "new" job tasks were discovered causing upward adjustments in the project's scope as measured in man-days. As is typically the case, however, the man-day adjustments made were not quite enough. This created a deficit in the project's man-day allocation which only became visible towards the end of the development phase when the development work was almost finished and the man-day budget was almost used up.

As the man-day deficit became visible the project's team reacted by working harder and longer hours in an attempt to bring the project back on track. This translates in the model into the higher values of the "ACTUAL FRACTION OF A MAN-DAY ON PROJECT" as shown in Figure 6.

But as was explained above, project teams are not, in general, willing to maintain an above-normal work rate indefinitely. On project DE-A this is exactly what happened. That is, the persistence of the work backlog eventually overwhelms the workforce's intensified efforts, and around day 300 arrangements were made with project management to handle the remaining project deficit through adjustments to both the project's man-day budget and its schedule.

The Control Subsystem

Decisions made in any organizational setting are based on what information is actually available to the decision maker(s). Often, this available information is inaccurate, i.e., not identical to the primary variables it represents. The information may be late, biased, and noisy. Apparent conditions may, therefore, be actually far removed from the actual

or true state, depending on the information flows that are being used and the amount of time lag and distortion in these information flows.

True productivity of a software project team is a good example of a variable that is often not knowable by members of the project. To know what the true value of productivity is at any point in time in the project, one needs to know the true values of both the total amount of project work accomplished to date and the resources expended. This, however, can be difficult to evaluate because the software product remains largely intangible during most of the development process.

How, then, is progress measured in a software project? Our own field study findings corroborate those reported in the literature, namely, that progress, especially in the earlier phases of software development, is measured by the rate of expenditure of resources rather than by some count of accomplishments. For example, a project for which a total of 100 man-days is budgeted would be perceived as being 10% complete when 10 man-days are expended; and when 50 man-days are expended it would be perceived as 50% complete, etc.

> 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 (DeMarco, 1982).

This surrogate for measuring project progress has some interesting implications on how management assesses the project team's productivity. When progress in the earlier phases of software development (call it time period t1), is measured by the rate of expenditure of resources, status reporting

ends up being nothing more than an echo of the original plan. That is, "MAN-DAYS PERCEIVED STILL NEEDED FOR NEW TASKS" (MDPNNT) becomes, under such conditions, simply equal to the "MAN-DAYS PERCEIVED REMAINING FOR NEW TASKS" (MDPRNT):

$$MDPRNT_{t1} = MDPNNT_{t1}$$

But "MAN-DAYS PERCEIVED STILL NEEDED FOR NEW TASKS" (MDPNNT) is (implicitly if not explicitly) equal to the value of "TASKS PERCEIVED REMAINING" (TSKPRM) divided by the manager's notion of the team's productivity, i.e., by the value of "PERCEIVED DEVELOPMENT PRODUCTIVITY" (PRDPRD). That is,

$$MDPNNT_{t1} = TSKPRM_{t1} / PRDPRD_{t1}$$

Substituting MDPRNT for MDPNNT, we get

$$MDPRNT_{t1} = TSKPRM_{t1} / PRDPRD_{t1}$$

which leads to,

$$PRDPRD_{t1} = TSKPRM_{t1} / MDPRNT_{t1}$$

This is an interesting result. For, it suggests that as project members measure progress by the rate of expenditure of resources, they, by so doing, would be implicitly assuming that their productivity equals "TASKS PERCEIVED REMAINING" (TSKPRM) divided by the "MAN-DAYS PERCEIVED REMAINING FOR NEW TASKS" (MDPRNT). What makes this interesting is the fact that such an assumed value for productivity is solely a function of future projections (i.e., remaining tasks and remaining man-days) as opposed to being a reflection of accomplishments (i.e., completed tasks and expended resources).

This implicit notion of productivity is captured in the model by the variable "PROJECTED DEVELOPMENT PRODUCTIVITY" (PJDPRD), defined as,

$$PJDPRD = TSKPRM / MDPRNT$$

As the project advances towards its final stages, and accomplishments

become relatively more visible, project members become increasingly more able to perceive how productive the workforce has actually been. As a result, perceived productivity ceases to be a function of projected productivity and is determined instead on the basis of actual accomplishments. That is, "PERCEIVED DEVELOPMENT PRODUCTIVITY" approaches the value of the project team's "ACTUAL DEVELOPMENT PRODUCTIVITY" (ACTPRD), i.e., the value of "CUMULATIVE TASKS DEVELOPED" (CUMTKD) divided by the value of "CUMULATIVE MAN-DAYS EXPENDED" (CUMDMD).

Thus in the final stages of a software project (call it time period t2),

$$PRDPRD_{t2} \text{ -----) } ACTPRD_{t2}$$

where,

$$ACTPRD = CUMTKD / CUMDMD$$

To recapitulate, in the early phases of development, "PERCEIVED DEVELOPMENT PRODUCTIVITY" is implicitly determined on the basis of future projections (i.e., remaining tasks and man-days). Towards the end of the project, on the other hand, "PERCEIVED DEVELOPMENT PRODUCTIVITY" gets to be explicitly determined on the basis of actual accomplishments. People's assumptions about their productivity, therefore, change as the project progresses. The change, however, is typically gradual not abrupt.

One "infamous" consequence of this error in perception deserves some discussion. It is the "90% syndrome phenomenon." Baber (1982) provides the following description of the problem:

> ... estimates of the fraction of the work completed (increase) as originally planned until a level of about 80-90% is reached. the programmer's individual estimates then increase only very slowly until the task is actually completed.

There is ample evidence in the literature on the pervasiveness of the 90% syndrome in software development projects (DeMarco, 1982). Its manifestation

in the simulated DE-A project is depicted in Figure 8. By measuring progress in the earlier phases of the project by the rate of expenditure of resources, status reporting ended up being nothing more than an echo of the project's plan. This created the "illusion" that the project was right on target. However, as the project approached its final stages (e.g., when 80-90% of the resources are consumed), discrepancies between the percent of tasks accomplished and the percent of resources expended became increasingly more apparent. At the same time, project members became increasingly able to perceive how productive the workforce has actually been. This results in a better appreciation of the amount of effort actually remaining. As this appreciation developed, it started to, in effect, discount the project's progress rate. Thus, although the project members proceeded towards the final stages of the project at a high work rate because of schedule pressures, their net progress rate slowed down considerably. This continued until the project completed.

The Planning Subsystem

In the Planning Subsystem, initial project estimates (e.g., for completion time, staffing load, man-days) are made at the beginning of the project. These estimates are then revised, as necessary, throughout the project's life. For example, to handle a project that is perceived to be behind schedule, plans can be revised to add more people, extend the schedule, or do a little of both. The Planning Subsystem is depicted in Figure 9.

By dividing the value of "MAN-DAYS REMAINING" (a measure of the magnitude of the effort still remaining) at any point in the project, by the "TIME REMAINING" a manager can determine the "INDICATED WORKFORCE LEVEL." This would represent the workforce size believed to be necessary and sufficient to

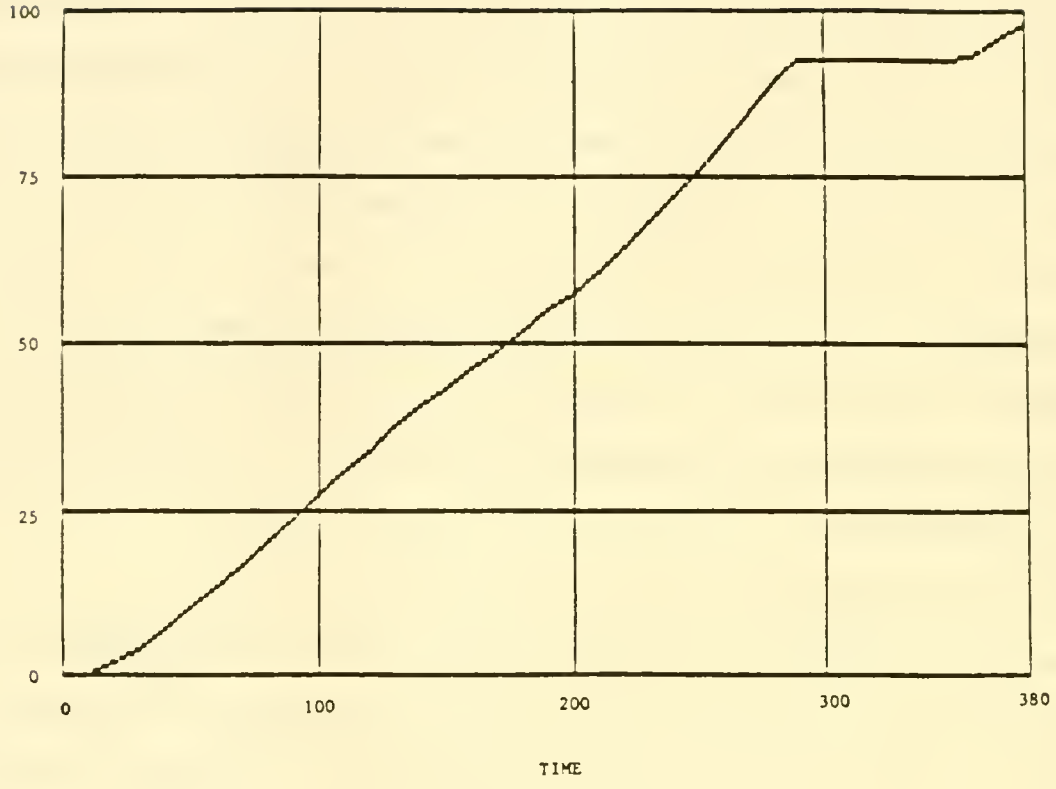


FIGURE 8

REPORTED % OF WORK COMPLETED OVER TIME

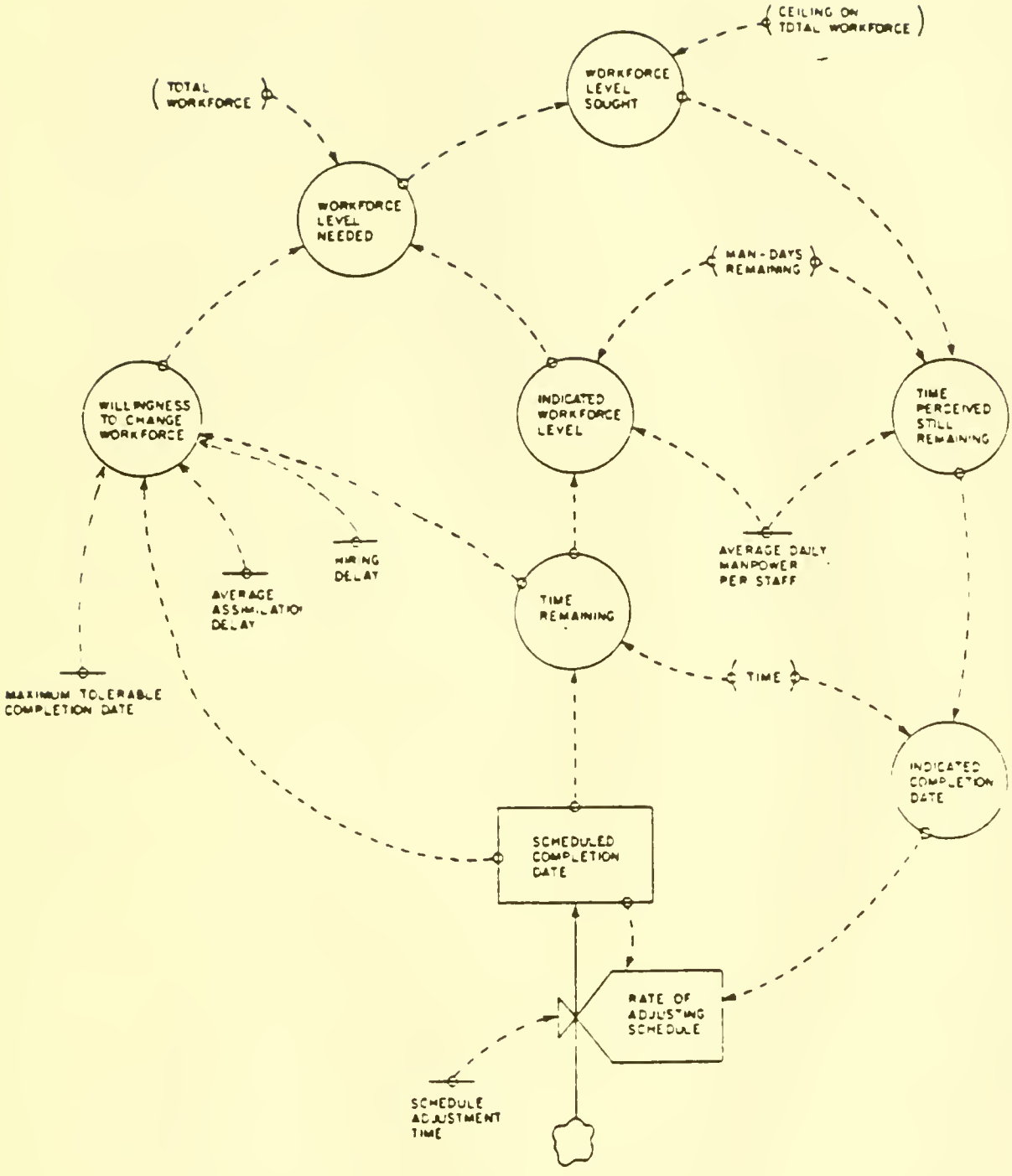


Figure 9

PLANNING SUBSYSTEM

complete the project on time based on the currently scheduled completion date. If this indicated workforce size turns out to be lower than the value of the actual workforce level on the project, excessive employees would simply be transferred out of the project. If on the other hand, the opposite is true, then this would indicate a need to add more people.

However, as has been explained in the Human Resource Management Subsystem, hiring decisions are not determined only on the basis of scheduling considerations. Consideration is typically given to the stability of the workforce. Different organizations weigh this factor to various extents. For example, workforce stability had relatively little weight in NASA's DE-A project, as we saw. Such organizational differences are captured in the model by the policy variable termed "WILLINGNESS TO CHANGE WORKFORCE LEVEL," and whose form is organization-specific.

By dividing the value of the "WORKFORCE LEVEL SOUGHT" (that emerges after the above set of factors is contemplated) into the value of the "MAN-DAYS REMAINING," management determines the time it perceives to be still required to complete the project. Once this, in turn, is known, it can be used to adjust the project's "SCHEDULED COMPLETION DATE," if necessary.

Referring back to Figure 6, notice how project DE-A's management was inclined <u>not</u> to adjust the project's scheduled completion date during most of the development phase of the project. Adjustments, in the earlier phases of the project, were instead made to the project's workforce level. This behavior is not atypical. It arises, according to DeMarco (1982) because of political reasons:

> Once an original estimate is made, it's all too tempting to pass up subsequent opportunities to estimate by simple 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.

With this discussion of the model's Planning Subsystem we conclude our overview presentation of the model's structure and behavior. For the interested reader, a more detailed description of the model's structure, its mathematical formulation, and of its validation is provided in [(Abdel-Hamid, 1984) and (Abdel-Hamid and Madnick, 1987a)].

The Model as an Experimentation Vehicle

Many authors have argued for the desirability of having a laboratory tool for testing ideas and hypotheses in software engineering (Thayer, 1979). For example, Weiss (1979) commented that in software engineering it is remarkably easy to propose hypotheses and remarkably difficult to test them.

The computer simulation tools of System Dynamics provide us with such an experimentation vehicle to test ideas and hypotheses relating to the software project management process. 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... Internally, the model provides complete control of the system's organizational structure, its policies, and its sensitivities to various events (Forrester, 1961).

Currently, the model is being used as an experimentaion vehicle to study and predict the dynamic implications of managerial policies and procedures on the software development process in areas such as: scheduling, control, quality assurance, and staffing. Three types of results are:

1. Uncovering dysfunctional consequences of some currently adopted policies. In (Abdel-Hamid and Madnick, 1986) we investigated the project scheduling practices in a major U.S. minicomputer manufacturer. In the particular organization, software project managers use Boehm's (1981) COCOMO model to come up with initial project estimates, which are then adjusted

upwards using a judgmental "safety factor" to come up with the project estimates actually used. The purpose of the experiment was to investigate the implications of this safety factor policy.

The results demonstrated how "a different schedule creates a different project." That is, if a software project were conducted twice using two different initial schedule estimations, the outcomes would be significantly different in nature (e.g., in terms of the actual completion time, workforce pattern, productivity). This is because a project's schedule creates pressures and perceptions that significantly affect how people behave on the project. The implications of this are:

- o. A more accurate estimate is not necessarily a better estimate. An estimation method should be judged not only on how accurate it is, but, in addition, it needs to be judged on how costly the projects it "creates" are.
- o. It is clear that both the software manager as well as the student of software estimation should reject the notion that a new software estimation tool can be adequately judged strictly on the basis of how accurately it estimates historical projects.

2. Providing support for managerial decision making. In (Abdel-Hamid and Madnick, 1987b) we reported on how the model is used to analyze the economics of the Quality Assurance (QA) function and to support the selection of the optimal investment level in QA.

3. Providing new insights into software project phenomena. One such phenomenon we examined is "Brooks' Law" (Abdel-Hamid, 1987). Brooks' Law states that adding manpower to a late software project makes it later (Brooks, 1978). Since its "enactment," Brooks' Law has been widely endorsed

in the literature. Furthermore, it has often been endorsed indiscriminately for systems programming-type projects and applications-type projects, both large and small. Interestingly, this wide-spread endorsement of Brooks' Law has taken place, even though the "Law" has not been formally tested.

Our objective in this experiment was to investigate the applicability of Brooks' law to the environment of medium-sized application-type software projects. The experimental results showed that while adding more people to a late project of this type does cause it to become more costly, it does not always cause it to complete later. The increase in the cost of the project is caused by the increased training and communication overheads, which in effect decrease the average productivity of the workforce and thus increase the project's cost in man-days. For the project's schedule to also suffer, the drop in productivity must be severe enough and late enough in the project's lifecycle to render an additional person's net cumulative contribution to the project to be, in effect, a negative contribution. Our experimental results indicate that this happens only under extremely aggressive manpower acquisition policies and where management's willingness to add new staff members persists until the very final stages of the testing phase.

Summary:

The tremendous growth in the demand for software systems over the last two decades has not been, for many organizations, a painless one. The record shows that the development of software has been marked by cost overruns, late deliveries, poor reliability, and users' dissatisfaction. Some refer to this set of difficulties as the "software crisis." These problems persist in spite of significant software engineering advances made over the last decade to tackle the technical hurdles of software production.

In recent years, the managerial aspect of software development has gained

recognition as being at the core of both the problem and the solution. Along with this recognition, though, serious and legitimate concerns have been raised. Chief among them is the belief that as of yet we still lack a fundamental understanding of the software development process, and that without such an understanding the likelihood of any significant gains on the managerial front is questionable.

The objective of the research project reported in this paper is to enhance our understanding of and gain insight into the general process by which software development in managed. To achieve this, we developed an integrative System Dynamics model of the software development process. The model complements and builds upon current research efforts, which tend to focus on the micro-components (e.g., scheduling, programming, productivity) by integrating our knowledge of these micro-components into an integrated continuous view of the software development process. An overview of the four subsystems of the model, namely, (1) human resource management, (2) planning, (3) control, and (4) software production was presented and discussed.

The model is being used as an experimentation vehicle to study and predict the dynamic implications of an array of managerial policies and procedures pertaining to the management of software projects. Important results have been obtained in areas such as understanding the impact of schedule estimation and manpower adjustments on actual project performance.

APPENDIX

The DE-A software project was conducted at the Systems Development Section of NASA's Goddard Space Flight Center (GSFC) at Greenbelt, Maryland in the 1979/1980 time period. The basic requirements for the project were to design, implement, and test a software system that would process telemetry data and would provide attitude determination and control for NASA's DE-A satellite.

The development and target operations machines were the IBM S/360-95 and -75. The programming language was mostly FORTRAN. Other project statistics include:

- Project Size (in Delivered Source Instructions) 24,000 DSI

- Cost (for design through system testing)

initial estimate	1,100 man-days
actual	2,220 man-days

- completion time (in working days)

intial estimate	320 days
actual	380 days

BIBLIOGRAPHY

1. Abdel-Hamid, T.K. "The Dynamics of Software Development Project Management: An Integrative System Dynamics Perspective." Unpublished Ph.D. dissertation, Sloan School of Management, MIT, January, 1984.

2. Abdel-Hamid, T.K. "The Dynamics of Software Project Staffing: A System Dynamics Based Simulation Approach." Accepted for publication in IEEE Transactions on Software Engineering, 1987.

3. Abdel-Hamid, T.K. and Madnick, S.E. "Impact of Schedule Estimation on Software Project Behavior." IEEE Software, (July, 1986).

4. Abdel-Hamid, T.K. and Madnick, S.E. Software Project Management. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., to be published in 1987a.

5. Abdel-Hamid, T.K. and Madnick, S.E. "The Economics of Software Quality Assurance: A System Dynamics Based Simulation Approach." Accepted for Publication in The Annals of the Society of Logistics Engineers, 1987b.

6. Baber, R.L. Software Reflected. New York: North Holland Publishing Company, 1982.

7. Basili, V.R. "Improving Methodology and Productivity through Practical Measurement." A lecture at the Wang Institute of Graduate Studies, Lowell, Mass., Nov., 1982.

8. Beck, L.L. and Perkins, T.E. "A Survey of Software Engineering Practice: Tools, Methods, and Results." IEEE Transactions on Software Engineering, Vol. SE-9, No. 5, (Sept., 1983).

9. Boehm, B.W. Software Engineering Economics, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1981.

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

11. Buckley, F. and Poston, R. "Software Quality Assurance." IEEE Trans. on Software Engineering, (January, 1984), 36-41.

12. Chrysler, E. "Some Basic Determinants of Computer Programmer Productivity." Communications of the ACM, Vol. 21, No. 6, (June, 1978), 472-483.

13. Cougar, J.D. and Zawacki, R.A. Motivating and Managing Computer Personnel. New York: John Wiley & Sons, Inc., 1980.

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

15. DePree, R.W. "The Long and Short of Schedules." Datamation, (June 15, 1984), 131-134.

16. Forrester, J.W. Industrial Dynamics. Cambridge, Mass: The MIT Press, 1961.

17. Freedland, M. "What You Should Know About Programmers." Datamation, March 15, 1987, pp. 91-102.

18. Ibrahim, R.L. "Software Development Information System." Journal of Systems Management, (Dec., 1978), 34-39.

19. Jensen, R.W. and Tonies, C.C. Software Engineering. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1979.

20. McKeen, J.D. "Successful Development Strategies for Business Application Systems." MIS Quarterly, Sept., 1983.

21. Mills, H.D. Software Productivity. Canada: Little, Brown & Co., 1983.

22. Musa, J.D. "Software Engineering: The Future of A Profession." IEEE Software, January, 1985, 55-62.

23. NASA. "Software Development History for Dynamics Explorer (DE) Attitude Ground Support System (AGSS)." GSFC Code 580, June, 1983.

24. Newport, John P. Jr. "A Growing Gap in Software." Fortune, April 28, 1986, 132-142.

25. Pooch, U.W. and Gehring, P.F. Jr. "Toward a Management Philosophy for Software Development." In Advances in Computer Programming Management. Edited by T.A. Rullo. Philadelphia, PA: Heyden & Sons, Inc., 1980.

26. Radice, A. "Productivity Measures in Software." The Economics of Information Processing: Operations, Programming, and Software Models. Volume II. Edited by R. Goldberg and H. Lorin. New york: John Wiley & Sons, Inc., 1982.

27. Ramamoorthy, C.V. et al. "Software Engineering: Problems and Perspectives." Computer, Oct., 1984.

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

29. Roberts, E.B. (ed.). Managerial Applications of System Dynamics. Cambridge, Mass: The MIT Press, 1981.

30. Shell, R.L. "Work Measurement for Computer Programming Operations." Industrial Engineering, (Oct., 1972), 32-36.

31. Shooman, M.L. Software Engineering - Design, Reliability and Management. New York: McGraw-Hill, Inc., 1983.

32. Steiner, I.D. Group Process and Productivity. New York: Academic Press, 1972.

33. Tausworthe, R.C. Standardized Development of Computer Software. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1977.

34. Thayer, R.H. "Modeling a Software Engineering Project Managaement System." Unpublished Ph.D. dissertation, University of California, Santa Barbara, 1979.

35. Thayer, R.H. et al. "Major Issues in Software Engineering Project Management." IEEE Transactions on Software Engineering, Vol. SE-7, No. 4, July, 1981.

36. Thomsett, R. People Project Management. New York: Yourdon Press, Inc., 1980.

37. Weick, K.E. The Social Psychology of Organization. 2nd edition. Reading, Mass: Addison-Wesley Publishing Co., Inc., 1979.

38. Weinberg, G.M. Understanding the Professional Programmer. Boston: Little, Brown & Co., Inc., 1982.

39. Weiss, D.M. "Evaluating Software Development by Error Analysis." Journal of Systems and Software, Vol. 1, 1979, 57-70.

40. Zelkowitz, M.V. "Perspectives on Software Engineering." Computing Surveys, Vol. 10, No. 2, (June, 1978).

41. Zmud, R.W. "Management of Large Software Development Efforts." MIS Quarterly, Vol. 4, No. 2, June 1980, 45-56.