AN INTEGRATIVE APPROACH TO MODELING THE SOFTWARE MANAGEMENT PROCESS: A BASIS FOR IDENTIFYING PROBLEMS AND EVALUATING TOOLS AND TECHNIQUES

Tarek K. Abdel-Hamid
Stuart E. Madnick

October 1982

CISR WP #95
Sloan WP #1366-82

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

AN INTEGRATIVE APPROACH TO MODELING THE SOFTWARE MANAGEMENT PROCESS: A BASIS FOR IDENTIFYING PROBLEMS AND EVALUATING TOOLS AND TECHNIQUES

Tarek K. Abdel-Hamid
Stuart E. Madnick

October 1982

CISR WP #95
Sloan WP #1366-82

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

Submitted for publication in the Proceedings of the IEEE Workshop on Software Engineering Technology Transfer, April 25-27, 1983

I. INTRODUCTION: THE PROBLEM

The past two decades have witnessed the development of a large number of software engineering tools and techniques for improving the production of software systems. As a result of these developments, software project managers today have at their disposal an abundance of sophisticated tools that are potentially useful in helping them increase their efffectiveness. And the number of these techniques continues to increase each year.

Still, "most software projects fail" (McClure, 1981). Many organizations are finding that their people are still developing the same expensive, bug-ridden, unmaintainable software that they were developing before the new software engineering tools (e.g., structured programming) were introduced (Yourdon, 1979).

A question that has frequently been raised (and appropriately so) is: who is to blame? Does the fault lie in the tools themselves, or are the tool-users, especially management, the real culprit?

In the literature, there is an abundance of arguments on both sides of the issue. For example, Thayer (1979) argued that the problems we are facing in software production are, largely, due to a lack of effective tools (especially) in the area of software project management. On the other hand, Yourdon (1979) sees management as the real "villain." It is his opinion that "... management is to blame for the failure of the structured revolution." He feels that, in most companies, management did a poor job in selling the new techniques, in providing the necessary training, and in general in providing the needed support and follow-through. And as a result, some argue, most of the software engineering tools, techniques, and methodologies that have been available for practical use for a long time, languish on the shelf like a good product which does not sell.

While there is certainly some validity in both arguments, we feel that the "true" answer lies somewhat between the two.

What we feel is still missing and much needed is not necessarily a set of new specific software engineering tools, nor a new breed of "super-capable" software project managers (and which is probably infeasible anyway), but rather a much needed model, perspective if you will, of software development project management, that can help both managers and researchers to better decide when, where, and how to use (or not use) the ever increasing number of software-engineering tools and techniques that are already available. It is interesting that (even) more

than a decade ago Aaron (1970) commented that "We ran into problems because we didn't know how to manage what we had, not because we lacked the techniques themselves."

Today's software development project managers are faced with a general situation that has been continuously becoming more complex (Singer, 1982). Their software development organizations develop new products, offer new services, incorporate additional technologies, and have a more heterogeneous workforce. This complexity often makes it less and less obvious how healthy or sick their organizations actually are. It also makes it less obvious how important various known problems are, and what the second- and third-order consequences of some set of actions --- such as the use of some software engineering tool --- will be.

The consequenses of this situation are predictable (Kotter, 1978):

> Since they lack confidence in their assessment of the risks and benefits of organizational improvement techniques, managers quite often choose not to use them. As a result, many potentially useful techniques are seriously underutilized. Even when they are used, they are sometimes used inappropriately. Managers select the wrong techniques, or use them at the wrong time or in the wrong way. Then, when their expectations are not fulfilled, they tend to become even less willing to experiment with organizational improvement tools.

Our objective in this research effort is provide both software development managers and researchers (not with yet another specific software engineering tool, but instead) with a useful way of thinking about organizational improvement issues.

Our aim is to develop an integrative model of software project management that can help them answer the difficult questions they need to raise when assessing organizational health, selecting improvement tools (from the many that are already available), and implementing their choices.

II. AN INTEGRATIVE SYSTEM DYNAMICS COMPUTER MODEL OF SOFTWARE PROJECT MANAGEMENT

In a special issue of the IEEE Transactions on Software Engineering on Project Management, Merwin (1978) asserted that:

> What is still needed is the overall management fabric which allows the senior project manager to understand and lead major data processing development efforts.

At MIT's Center for Information Systems Research (CISR), we are currently engaged in a research project to develop such an "overall management fabric." Specifically, our objective is to develop an integrative system dynamics computer model of software development project management. Such a model (we feel) would be helpful to software development managers and researchers in handling the ever increasing complexities of software production, and thereby improve their abilities in identifying more accurately both their more important problems, as well as the more effective solutions to those problems.

Models --- which are usually simplified but formal abstractions of real-world systems --- have been effectively used, for many years now, by practitioners in fields like operations research, management science, and systems analysis to handle specific complex decision problems (Cleland and King, 1975). It is important to realize, here, that we, on the other hand, are attempting to develop a general-type of model. Can such a model, one might ask, have any widespread applicability?

In an empirical investigation of the objectives and constraints of EDP departments in various industries, Hallam (1975) found high goal congruence and high constraint congruence i.e., a high degree of agreement was found among all types of EDP departments studied regarding goals and constraints. And based on the findings of his study Hallam then concluded that:

> The primary beneficiary of the description here of EDP goals and constraints is the model builder interested in modeling the EDP management process. The agreement found among all types of EDP departments regarding goals and constraints should encourage model builders, since it indicates that a general EDP department model should have widespread applicability. (underlining ours)

In the remainder of this section, we would like now to argue for the three characteristic "features" of our model, which together differentiate our modeling approach from that of many others in the area of software engineering. The three characteristic features being: (1) it is an integrative model; (2) it is a computer model; and (3) it is a system dynamics model.

II.1. Why an Integrative Model:

Let us start off by first demonstrating that we are not (completely) alone in believing that building an integrative model of software development project management is not an infeasible venture:

> I believe that by combining the various functions, actions, and interactions of software development management and overlaying them on a framework of a standard management model, a model of a generalized software engineering project management system can be identified. (Thayer, 1979)

The integrative feature of our model should help software project managers in two important ways. First, it should help them diagnose more accurately what is causing and what has lead to whatever problems they have identified. It would do that, primarily, by "alerting" managers to all the relevant facets of software production e.g., human as well as technological.

Because "interactions and interdependencies are common in all social systems, and are major complicating factors which necessitate an overall system concept" (Cleland and King, 1975), one of the major difficulties facing both students of organizations and managers trying to improve their functioning is the lack of such overview models (Schein, 1980).

Many studies have indicated that managers often deal with the problems they encounter in terms of mental models that do not necessarily include all the elements or aspects of the problematic situation. Technically trained managers, in particular, tend to underestimate the influences of their internal social systems on organizational performance (Kotter, 1978). Consider, for example, the problem of achieving software reliability. By explicitly incorporating the managerial functions of planning and staffing together with the technical processes of software development (e.g., desining, coding, ... etc.) in an integrative model, a manager is "prompted" to investigate not only the technical issues of software reliability, but also the implications of:

* Pressures to begin coding before the design is completed because of tight schedules.
* Insufficient emphasis on programmer education and training.
* Poor matching of programmers' abilities with job assignments.

(In a study reported by Myers (1976), the above three factors contributed more to the generation of serious software errors than did any weaknesses in the design, implementation, or testing processes.)

The second way in which the integrative feature of our model should be helpful is that it would provide managers with a rational basis for identifying feasible "improvement interventions," and for assessing their probable impact once implemented.

Again, by providing a comprehensive world view, the model should help managers assess the second- and third-order consequences of some set of actions.

The chain of effects in going from a particular managerial intervention (e.g., hiring more people) to immediate consequences then to second- and third-order consequences and newly created problems is one of the pervasive characteristics of modern social systems. Quite literally, in such systems everything depends on everything else (Cleland and King, 1975). That is why, many researches assert that overview models can be major aids to managers who are trying to improve their organizations' effectiveness (Schein, 1980).

For example, the software project manager who is contemplating hiring more people to speed up a late project, would be "prompted" by our integrative model to investigate the dynamic implications of such a decision on things such as:

* the human communication overhead, and the effect of that on productivity, and
* the time and effort allocated by the experienced and productive team members to train new personnel.

II.2. Why a Computer Model

Using an integrative model merely to "alert" managers to all the important aspects of a problem, while clearly useful and essential, is definitely not enough. Because such a model will undoubtedly contain a large number of components with a complex network of interrelationships, we must <u>in addition</u> provide an effective means to determine both accurately and efficiently the dynamic behavior implied by such component interactions.

> Since the ultimate aim is to explain and predict the behavior of organizations, not of individual components, it is necessary to have a method which allows us to construct and manipulate a total organization. Computer simulation techniques provide one such method. (Cohen and Cyert, 1963)

Computer models have been, of course, widely used to simulate many of our complex technological systems. Our social systems are far more complex and harder to understand than our technological systems. Why, then, do we not use the same approach of making computer models of social systems and conducting "laboratory experiments" on these models before we try new policies and procedures?

> The answer is often stated that our knowledge of social systems is insufficient for constructing useful models. But what justification can there be for the apparent assumption that we do not know enough to construct models but believe we do know enough to directly design new social systems by passing laws and starting new social programs? I am suggesting that we now do know enough to make useful models of social systems. Conversely, we do not know enough to design the most effective social systems directly without first going through a model-building experimental phase. But I am confident, and substantial supporting evidence is beginning to accumulate, that the proper use of models of social systems can lead to far better systems, laws, and programs. (Forrester, 1971)

Experience from working with managers in many environments indicates that they are generally able to <u>specify</u> the detailed relationships and interactions among managerial policies, resources, and performance. However, managers are usually unable to determine accurately the dynamic behavior implied by these relationships. Human intuition, studies have shown, is illsuited for calculating the consequences of a large number of interactions over time (Richardson and Pugh, 1981).

Unlike a mental model, a system dynamics computer model can reliably and effeciently trace through time the implications of a messy maze of interactions. And it can do that without stumbling over phraseology, emotional bias, or gaps in intuition (Richardson and Pugh, 1981).

By utilizing computer simulation techniques in this research effort we, thus, combine the strengths of the manager with the strengths of the computer. The manager aids by specifying relationships within the software project managent system, the computer then calculates the dynamic consequences of these relationships.

II.3. Why a System Dynamics Model

System dynamics is the application of feedback control systems principles and techniques to managerial and organizational problems (Roberts, 1981).

Most succinctly, feedback is the transmission and return of information. The emphasis, inherent in the word feedback itself, is on the return. A feedback loop exists whenever an action taker will later be influenced by the consequences of his or her actions. Feedback loops divide naturely into two categories, which are labeled deviation-amplifying feedback (DAF) or positive loops, and deviation-counteracting feedback (DCF) or negative loops.

It is pertinent that we think in terms of feedback loops because (Weick, 1979):

> 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, ...

A point which is important in particular to the application of deviation-amplifying feedback (DAF) to management, concerns the distinction between (1) the initial event (from outside a loop) which starts the deviation amplifying process in motion, and (2) the dynamics of the feedback process which perpetuates it. While the initial event is important in determining the direction of the subsequent deviation amplification, the feedback process is <u>more important to an understanding of the system</u> (Ashton, 1976). The initial event sets in motion a cumulative process which can have final effects quite out of proportion to the magnitude of the original push. The push might even be withdrawn after a time, and still a permanent change will remain or even the process of change will continue without a new balance in sight. A further problem is that, after some period of time has elapsed, it may be difficult, if not impossible, to discover the initial event. An interesting example of this has been provided by Wender (1968):

> ... a fat and pimply adolescent may withdraw in embarrassement and fail to acquire social skills; in adulthood, acne and obesity may have disappeared but low self-esteem, withdrawal, and social ineptitude may remain. Social withdrawal and low self esteem are apt to stay fixed because the DAF chain <u>now</u> operates: social ineptitude leads to rejection, which leads to lowered self-esteem, greater withdrawal, less social experience, and greater ineptitude. <u>What has initiated the problem is no longer sustaining it.</u> A knowledge of the problem's origin would not be expected to alter the <u>currently operative</u> loop unless such insight served to motivate behavioral change ...
> Finding the initial event (acne and obesity) may have less usefulness than understanding the current sustaining feedback mechanism. Furthermore, in some instances the initial event may have left no traces of its existence and may be undiscoverable.

It is no wonder, then, that "most mangers get into trouble because they forget to think in circles. I mean this literally. Managerial problems persist because managers continue to believe that there are such things as unilateral causation, independent and dependent variables, origins, and terminations" (Weick, 1979).

III. AN EXAMPLE APPLICATION OF THE MODEL

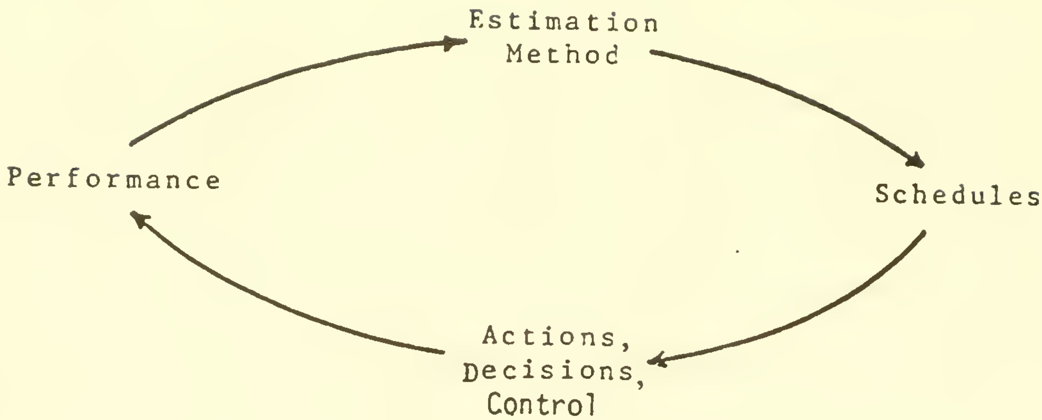
A first version of our integrative system dynamics computer model of software development project management has already been developed. The model is presented and discussed in (Abdel-Hamid and Madnick, 1982a). And in (Abdel-Hamid and Madnick, 1982b) the model was used to investigate the dynamics of software project scheduling.

It would be useful to conclude this report with a quick discussion of the results of our second paper, as this would put the concepts and ideas we've been discussing so far in the more perceptible form of a specific example application.

As mentioned above the specific problem area studied was that of software project scheduling. The software industry is young, growing, and marked by rapid change in technology and application. It is not surprising, then, that the ability to estimate project resources (including the time resource) is still relatively undeveloped. In the last few years there has been a surge in activity to develop quantitative resource estimation methods e.g., TRW's COCOMO model (Boehm, 1981) and Putnam's SLIM model (Putnam, 1980). Because such currently available quantitative techniques are first, usually tailored to a limited set of project/organizational types, and second, are (still) imperfect, the developers of such techniques emphasize the necessity to continuously collect project data via the planning and control activities, compare estimates to actuals, and use the results to "tune" the estimating tools (Boehm, 1981).

Meanwhile research findings over the past few years have clearly shown that the decisions that people make in organizations, and the actions they choose to take are <u>significantly</u> influenced by the pressures, perceptions, and incentives produced by the organization's planning and control system(s) (Weil, 1981). In particular, knowledge of <u>project schedules</u> was found to affect the real progress rate that is achieved, as well as the progress and problems that are reported upward in the organization.

What this implies is the existence of a feedback loop (see below), whereby an estimation technique produces project schedules, which affect the decisions and actions of the technical performers and their managers, this in turn affects work performance, which eventually is fed into the organization's projects' database to influence future estimations.



Notice that the above <u>feedback</u> loop <u>integrates</u> many different aspects of software development. It integrates technical as well as human aspects, planning as well as control aspects, and managerial as well as production aspects. Such an integrative system dynamics perspective of the scheduling problem, it is important to realize, is quite different from the more common and more limited perspective which views the scheduling problem as merely that of producing "better" estimates.

But what does the existence of such a feedback loop mean? Is it good or bad?

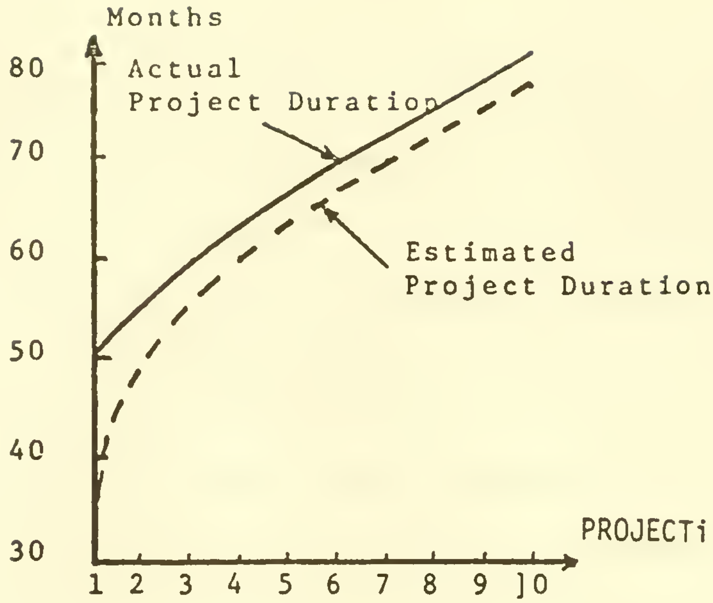
To most of us, the answers to such questions will <u>not</u> be intuitively obvious i.e., we can not answer them (with confidence) merely on the basis of our private mental models. The human mind is not adapted to correctly anticipate the dynamic consequences of interactions between the parts of a complex social system (Forrester, 1971), such as that of software project management.

Unlike a mental model, a <u>computer</u> model can reliably trace through time the implications of a messy maze of interactions.

Our computer model was thus utilized to conduct a "laboratory experiment" to investigate the implications of the above feedback loop. The experiment involved a hypothetical situation in which a company undertakes a sequence of ten software projects of identical size. It is assumed that an estimation tool is used in scheduling the projects. After each project is completed, its statistics (e.g., size, total time, ... etc.) are fed into an "experience database" and used to "tune" the estimation tool. Once tuned, it is then used to estimate the next project, and so on.

After the experiment was completed i.e., after running through the ten projects, we were surprised to observe that in all projects, the schedule was always overrun, as shown in the figure below. Notice that management started each project (e.g., "PROJECTi") with a slightly longer scheduled duration than the previous one (i.e., "PROJECTi-1"), and still "PROJECTi" would always overrun its schedule, causing management to use an even longer scheduled duration time for the next project (i.e., "PROJECTi+1"), and so on.

The (surprising) phenomenon we ecountered is one that has been frequently observed in system dynamics studies of social systems. It has been termed "The Policy Resistence of Social Systems," "Shifting the Burden to the Intervener," and "Addiction" among other things. While a full explanation is presented in (Abdel-Hamid and Madnick, 1982b) it suffices here to draw a simple



analogy to what was going on. And that is the (familiar) problem of caffeine addiction, whereby an addict has to consume a certain amount of caffeine per day to maintain a certain level of alertness. As time goes on, the burden of maintaining alertness will keep shifting from the normal physiological body processes to the externally supplied caffeine dose. The result, of course, is that higher and higher doses will be required to maintain the <u>same</u> level of alertness.

IV. CONCLUSION

This paper is a report on an ongoing research project at MIT's Center for Information Systems Research (CISR) to develop an integrative system dynamics computer model of software development project management. We feel that such a model can help software development managers and researchers answer the difficult questions they need to raise when assessing organizational health, selecting software engineering "improvement tools" (from the many that are already available), and in implementing their choices.

Our modeling approach is different from that of many others. In section (II) we argue for the attractiveness of the three characterisic "features" of our model, namely, that it is an (1) integrative, (2) computer, and (3) system dynamics model.

A first version of our model has already been developed and used. In section (III) we discuss some of the results of its first application, to the problem of software project scheduling.

Currently we are conducting field studies at a number of organizations that are involved in the production of software systems. The data gathered will be used to develop a second more detailed model. Two planned applications of the model will then follow. In the first, we will use the model to uncover any "people-management" problems, which the management of one organization "feel" are causing cost and schedule overruns. The second application of the model will be to evaluate the impact of a comprehensive (and expensive) project planning and control system that has been recently installed in another organization.

BIBLIOGRAPHY

1. Aaron, J. D. "The Super-Programmer Project." Software Engineering Techniques: Report on the 1969 Rome Conference. Editted by J. N. Buxton and B. Randell. Brussels, Belgium: NATO Science Committee, 1970.

2. Abdel-Hamid, T. K. and Madnick, S. E. "A Model of Software Project Management Dynamics." The Sixth Int'l Computer Software and Applications Conference (COMPSAC), November 8-12, 1982a.

3. Abdel-Hamid, T. K. and Madnick, S. E. "The Dynamics of Software Project Scheduling: A System Dynamis Perspective." The Third International Conference on Information Systems, December 13-15, 1982b. Also to appear in an early 1983 issue of the Communications of the ACM.

4. Ashton, R. H. "Deviation-Amplifying Feedback and Unintended Consequences of Management Accounting Systems." Accounting, Organization and Society, Vol. 1, No. 4 (1976).

5. Bacon, G. "Software." Science, February, 1982.

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

7. Cleland, D. I. and King, W. R. Systems Analysis and Project Management. New York: McGraw Hill, 1975.

8. Cohen, K. J. and Cyert, R. M. "Computer Models in Dynamic Economics." A Behavioral Theory of the Firm. By R. M. Cyert and J. G. March. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1963.

9. Cooper, J.D. "Corporate Level Software Management." IEEE Tr. on Software Management, SE-4. No. 4, (July, 1978).

10. Cooper, K. G. "Naval Ship Production: A Claim Settled and a Framework Built." Interfaces, Vol. 10, No. 6,

(Dec. 1980).

11. Forrester, J. W. "Counterintuitive Behavior of Social Systems." Technology Review, January, 1971.

12. Gehring, P. F. and Pooch, U. W. "Software

13. Hallam, S. F. "An Empirical Investigation of the Objectives and Constraints of Electronic Data Processing Departments." Academy of Management Journal, Vol. 18, No. 1, (March, 1875).

14. Kotter, J. P. Organizational Dynamics: Diagnosis and Intervention. Reading, Massachusetts: Addison-Wesley Publishing Company, 1978.

15. Lave, C. A. and March, J. G. An Introduction to models in the Social Sciences. New York: Harper & Row, 1975.

16. McClure, C. L. Managing Software Development and Maintenance. New York: Van Nostrand Reinhold Company, 1981.

17. Merwin, R. E. "Guest Editorial---Software Management." IEEE TR. on Software Engineering, SE-4, No. 4 (July, 1978).

18. Myers, G. J. Software Reliability. New York: John Wiley & Sons, 1976.

19. Powers, R. F. and Dickson, G. W. "Misproject Management: Myths, Opinions, and Reality." California Management Review, Spring, 1973.

20. Putnam, L. H. "Software Cost Estimation and Life-Cycle Control: Getting the Software Numbers," IEEE Computer Society, IEEE Catalog No. EHO 165-1, 1980.

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

22. Roberts, E. D. The Dynamics of Research and Development. New York: Harper & Row Publishers, 1964.

23. Roberts, E. B., ed. Managerial Applications of System Dynamics. Cambridge, Massachusetts: The MIT Press, 1981.

24. Schein, E. H. Organizational Psychology. 3rd edition. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1980.

25. Scott, R. F. and Simmons, D. B. "Predicting Programmers Group Productivity: A Communication Model." IEEE Tr. on Software Engineering, SE-1, No. 4, (Dec., 1975).

26. Singer, L. M. The Data Processing Manager's Survival Manual. New York: John Wiley & Sons, 1982.

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

28. Weick, K. E. The Social Psychology of Organizing. 2nd edition. Reading, Massachusetts: Addison-Wesley Publishing Company, 1979.

29. Weil, H. B. "Industrial Dynamics and Management Information Systems." Managerial Applications of System Dynamics. Edited by E. B. Roberts. Cambridge, Massachusetts: The MIT Press, 1981.

30. Wender, P. H. "Vicious and Virtuous Circles: The Role of Deviation-Amplifying Feedback in the origin and Perpetuation of Behavior." Psychiatry, Nov., 1968.

31. Yourdon, E. "The Second Structured Revolution." Software World, Vol. 12, No. 3, (1979).

Date Due
