A GENERIC SYSTEM DYNAMICS MODEL OF SOFTWARE PROJECT MANAGEMENT

BY

John D. W. Morecroft
Tarek K. Abdel-Hamid

Massachusetts Institute of Technology
Sloan School of Management
Cambridge, Massachusetts 02139

To be presented at the 1983 International System Dynamics Conference, Chestnut Hill, Massachusetts, July 27-30, 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 computer software systems. As a result of these developments, today's software project managers have at their disposal an abundance of sophisticated tools that are potentially useful in helping them increase their effectiveness. 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 (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 consequences 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 to provide both software development managers and researchers 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 MODELING APPROACH 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.

In the remainder of this section, we would like to describe three characteristic "features" of our model, which 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:

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., designing, 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 would be helpful is in providing managers with a rational basis for identifying feasible "improvement interventions," and for assessing their probable impact once implemented.

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 in addition 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)

Experience from working with managers in many environments indicates that they are generally able to specify 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 computer simulation model can reliably and efficiently 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 management 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).

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 to the application, in particular, 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 more important to an understanding of the system (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 now operates: social ineptitude leads to rejection, which leads to lowered self-esteem, greater withdrawal, less social experience, and greater ineptitude. What has initiated the problem is no longer sustaining it. A knowledge of the problem's origin would not be expected to alter the currently operative 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. MODEL STRUCTURE

In an empirical investigation of the objectives and constraints of EDP departments in various industries, Hallam (1975) found a high degree of agreement 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 a widespread applicability.

While encouraged by these results, we, however, needed to be cautious. Since our aim was to develop a generic model, not of the management of an EDP department, but , rather, of the management of a software project, we knew that the generalizability of our model could not cross project-type boundaries.

Recent research findings in software engineering indicate

that there are several modes of software development, and that there are significant differences between these modes. For example, "These software development modes have cost-estimating relationships which are similar in form, but which yield significantly different cost estimates for software products of the same size" (Boehm, 1981).

Boehm identified two main modes, and labelled them the "organic mode" and the "embedded mode." The most common mode of software development is the organic mode: the small-to-medium size product developed in an in-house, familiar, software development environment. In such an environment, most people connected with the project have extensive experience in working with related systems within the organization. A significant implication of this is that most of the project members can usefully contribute to the project in its early stages, without generating a great deal of project communications overhead in finding out what the project is all about and what everybody else is doing.

In contrast, an embedded mode project envolves the development of a large software product (e.g., > 100,000 lines of code) usually in an "uncharted" environment e.g., requiring innovative data processing architectures or algorithms. The major distinguishing factor of an embedded mode software project, however, is a need to operate within tight constraints. The product must operate within (is embedded in) a strongly coupled

complex of hardware, software, regulations, and operational procedures, such as an electronic funds transfer system or an air traffic control system. In general, the costs of changing the other parts of this complex are so high that their characteristics are considered essentially unchangeable, and the software is expected both to conform to their specifications, and to take up the slack on any unforseen difficulties encountered or changes required within the other parts of the complex.

As a result, the embedded mode project does not generally have the option of negotiating easier software changes and fixes by modifying the requirements and interface specifications. The project must, therefore, expend more effort in accomodating changes and fixes. It must also expend more effort in assuring that the software actually meets the specifications and in assuring that changes are made correctly. These factors contribute both to lower productivity and to greater diseconomies of scale on larger projects.

Our present model is that of organic-type software projects (the most common type), and is based on literature findings as well as field studies of such projects.

The information we gathered from an extensive survey of the software engineering literature was used to develop the model's "skeleton," i.e., the model's major components and the interrelationships between them. This effort was then

complemented by a series of field studies, which we conducted at two large U.S. corporations, a major auto maker and a leading mini-computer manufacturer.

The software production teams we studied were, in both organizations, involved in the development of software systems for in-house use. For example, one team in the auto maker was developing a corporate-wide transportation system to "trace and control the movement of finished vehicles, material shipments, and vehicles of conveyance such as railcars and trucks." Another team in the mini-computer manufacturer was busy developing an "Export Services Operations System," that should allow the company to accelerate the preparation of export liscence applications.

The software systems that "our" teams were involved with were medium in size i.e., involving teams of 5-10 members working for six months to two years on systems that were 10,000 to 80,000 lines of code.

III.1. The Model's Subsystems:

The model consists of three subsystems: (1) The Manpower Management Subsystem (MMS); (2) The Software Production Subsystem (SPS); and (3) The Planning and Control Subsystem (P&CS) as shown in Figure (1).

The Manpower Management Subsystem contains the manpower

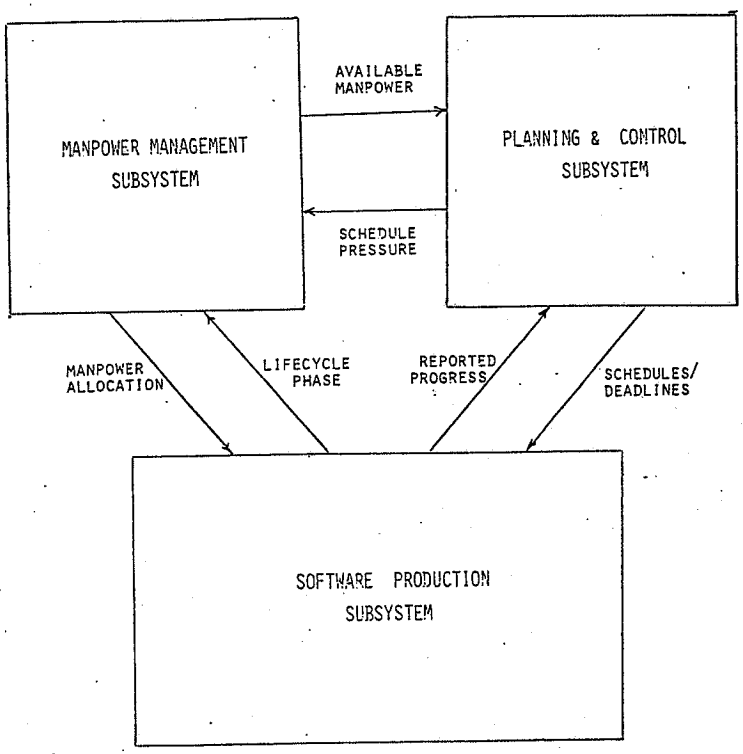

FIGURE (1): THE MODEL'S SUBSYSTEMS

allocation activities e.g., the allocation of manpower between quality assurance (QA) and coding at the coding phase of the project. The Software Production Subsystem captures the software production activities such as coding and testing. And finally, the Planning and Control Subsystem includes the managerial activities for assessing project status, projecting future manpower needs, and adjusting project schedules when necessary. Figure (1) also shows some of the important information connections which couple the subsystems together.

Each of the three subsystems is discussed in some detail next.

III.2. The Manpower Management Subsystem:

In both organizations, software professionals were allocated to only a single project at a time. Available manpower was, thus, a function of simply the size of the project team.

As shown in Figure (2), part of the available manpower on a project is allocated to Quality Assurance (QA) activities e.g., reviews, structured walkthroughs, ... etc. This effort is allocated throughout the life of the project based on the "QA POLICY." The total Man-Days allocated for this activity is usually between 5 and 10 percent of the total Man-Days for the project. As seen in Figure (2), Quality Assurance activities are affected by schedule pressures. When a software project falls

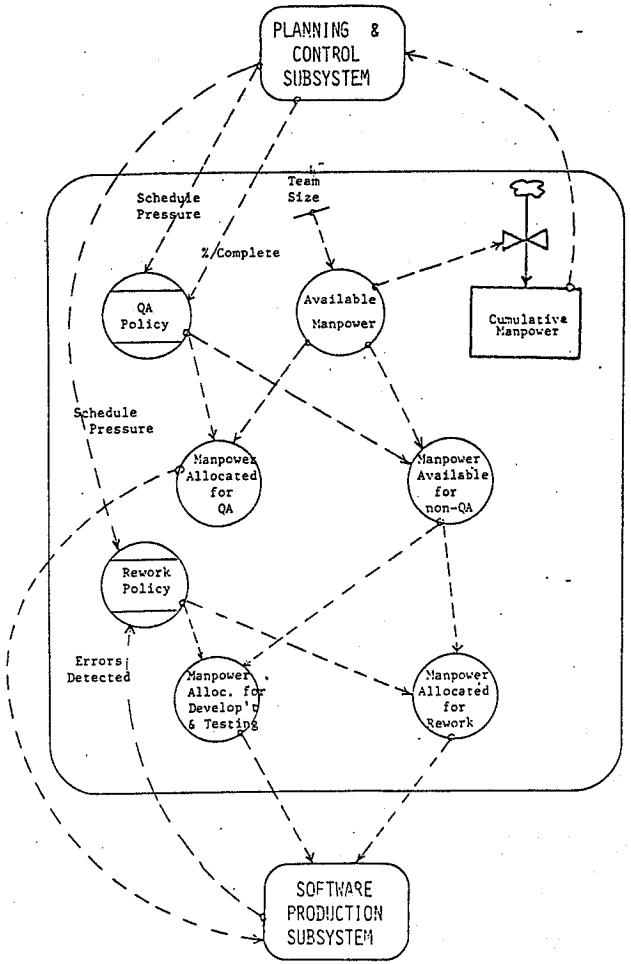

FIGURE (2): MANPOWER MANAGEMENT SUBSYSTEM

behind schedule, pressures from impatient users and frustrated managers often cause walkthroughs to be done hurriedly, without adequate preparation --- or not at all.

The objective of QA activities is, of course, to detect software errors as quickly as possible. Once erroneous tasks are detected, they need to be reworked. The manpower effort allocated to this activity will, obviously, be a function of the number of errors detected. It will also depend on how fast the errors need to be corrected. This delay is usually one week e.g., errors "caught" in this week's walkthrough need to be corrected by next week's walkthrough, when all corrections would be reviewed by fellow team members. The delay, however, usually increases as schedule pressures mount, since schedule pressures (as was mentioned above) often lead to less walkthroughs, with longer periods in-between. In most projects, reworking consumes between 5 and 10 percent of the total Man-Days of the project.

The remaining manpower effort is used to design, code and test the software system. Testing here refers to integration testing i.e., testing how the different software modules work together in an integerated system. This activity usually consumes the last 20 % of the software lifecycle.

III.3. The Software Production Subsystem:

A software system is defined in the model in terms of a

number of "tasks," where a task is 30 delivered source instructions (DSI) e.g., a one page subroutine. Progress in the project is made as workers perform tasks. The work rate is a function of the manpower allocated and the average productivity.

Software productivity is, of course, a function of a large number of factors e.g., system complexity, quality of personnel, ... etc. However, for a given project many of these factors remain constant and thus have no dynamic influence. In Figure (3) we depict two factors that impact software productivity during the course of a single project, namely, schedule pressures and learning.

Time pressures to meet deadlines has been found by Boehm (1981) and others, to increase software productivity. In our own field studies we found this to be true as long as team members perceive the schedule as attainable.

Studies have also shown that software productivity is affected by learning. As a project proceeds, the team members learn their job better, and this allows them to work faster. Mostly, the learning involves the application itself, but it may also extend to such things as new hardware, software, languages, ... etc.

Once tasks are worked, they are reviewed e.g., in a structured walkthrough. As was mentioned above, it is the

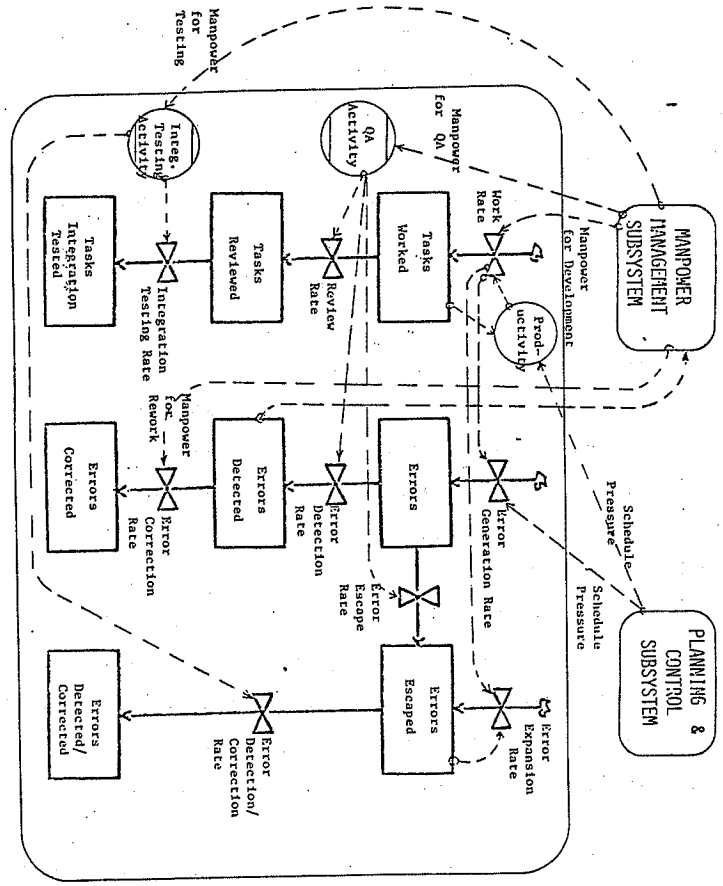

FIGURE (3): SOFTWARE PRODUCTION SUBSYSTEM

objective of the reviewing activity to "catch" software errors as early as possible. Errors that are detected will require correction, siphoning off manpower effort, that would otherwise be used to work on new tasks.

Unfortunately, our current software reviewing techinques are still imperfect ... they do fail to detect all errors. So, it is the objective of integration testing to detect and correct those errors that QA fails to "catch." Such errors, however, are very expensive. This is simply because these errors do not remain dormant awaiting detection and correction at the testing phase. They, instead, lead an "active existence" reproducing more and more errors. In fact, a study at TRW showed that it is 10 times more costly to fix a design error in the testing phase than it is at the design phase i.e., where it was "born" (Boehm, 1981).

III.4. The Planning and Control Subsystem:

As shown in Figure (4) the P&CS operates largely on the basis of progress information supplied by the Software Production Subsystem throughout the project's lifecycle.

Tasks worked are compared to the "perceived job size in tasks" to determine the "tasks perceived remaining." These are then translated into "Man-Days perceived still needed." If "Man-Days perceived still needed" is more than"Man-Days remaining," a schedule pressure develops that often leads to

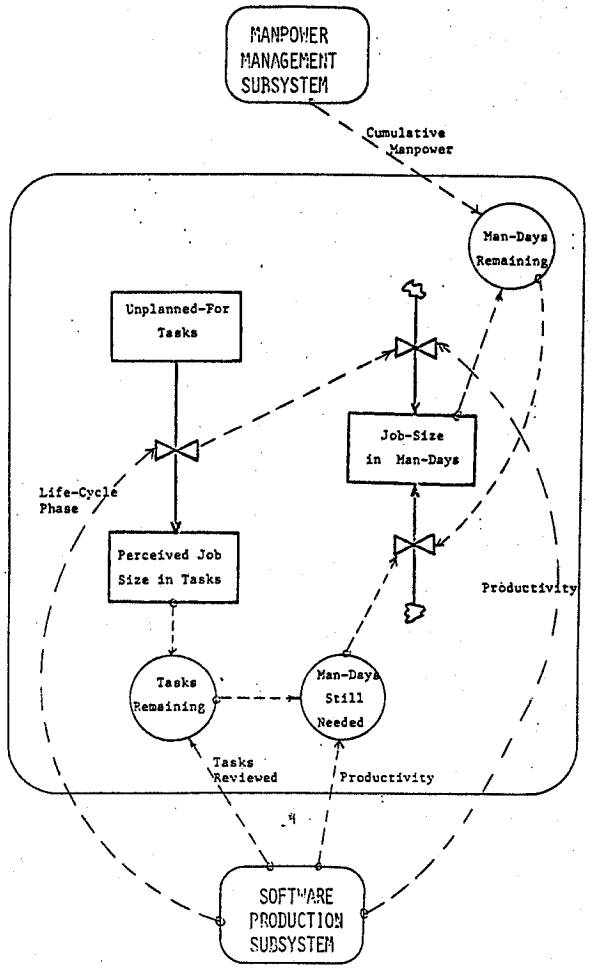

FIGURE (4): PLANNING & CONTROL SUBSYSTEM

longer working hours and higher productivity. Such a stressful situation can not continue for long, however. Eventually, project management finds it necessary to extend the schedule i.e., increase the "Job size in Man-Days."

One reason why projects tend to experience schedule problems is that project managers often under-estimate the size of the project. A common mistake is to underestimate the size of the support software. As the project proceeds, and the level of understanding increases (e.g., due to learning), such additional tasks are "discovered." This, thus, expands the perceived job size and creates schedule problems.

IV. FUTURE RESEARCH DIRECTIONS

Our overall goal in developing our integrative system dynamics computer model is to establish a basic understanding of software project management. In particular, we hope to build a tool that 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 system dynamics model has been coded and is running. We are currently testing/validating it. Once this is completed, we plan to use the model in two ways: Problem identification and evaluation of improvement interventions.

IV.1. Problem Identification:

There is an abundance of "suggested" problems in software production. Among the "front runners:" poor resource estimation,

changes in requirements, high personnel turnover, lack of progress visibility, poor life cycle balancing, ... etc.

Our model can play two useful roles. First, it can be used in testing the seriousness of the above problems. A similar SD application was reported by Cooper (1980) in the shipbuilding industry where the cost of changes in user requirements was evaluated.

The second possible role of the model would be to uncover still uncharted problem areas. Lave and March (1975) wrote that "A beautiful model is unpredictable." We feel that studying the interactions of factors such as those mentioned above can yield new insights into the causes of software project failures. For example, in (Abdel-Hamid and Madnick, 1982b) we "discovered" that the interactions of hiring/firing policies together with a high personnel turnover rate contributed more to the scheduling problem than did the estimation procedure.

IV.2. Evaluation of "Improvement Interventions:"

The utilization of organizational improvement techniques generally requires the expenditure of some scarce resource e.g., money, time, ... etc (Kotter, 1978). Because of this fact, and because of the large number of tools and techniques that have been developed, managers today have far more options for improving their organizations' effectiveness than they have resources. Our

model would provide managers with a useful way to think about organizational improvement issues. It would, for example, allow managers to experiment with new policies at a cost significantly lower than that of real-life experiments (Roberts, 1981).

In particular, we propose to investigate the implication of a new project planning and control system recently installed in one organization. Project planning and control systems are interesting to investigate because so much confusion surounds them. For example, Boehm (1981), based on TRW's experiences, strongly recommends them; Thayer (1979) found them to have no impact, one way or the other, on projects' success or failure; and Powers and Dickson (1973) found them to be dysfunctional.

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).