WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

# ON THE PORTABILITY OF QUANTITATIVE SOFTWARE ESTIMATION MODELS

Tarek K. Abdel-Hamid

Stuart E. Madnick

January 1988 #WP 1977-88

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

# ON THE PORTABILITY OF QUANTITATIVE SOFTWARE ESTIMATION MODELS

Tarek K. Abdel-Hamid

Stuart E. Madnick

January 1988 #WP 1977-88

# ON THE PORTABILITY OF QUANTITATIVE SOFTWARE ESTIMATION MODELS

By

Tarek K. Abdel-Hamid
Administrative Sciences Department
Naval Postgraduate School
Moterey, California 93943

Stuart E. Madnick
Sloan School of Management
Massachusetts Institute of Technology
50 Memorial Drive
Cambridge, Massachusetts 02139

Paper Category: Research

# ON THE PORTABILITY OF QUANTITATIVE SOFTWARE ESTIMATION MODELS


## Abstract

Evidence in the literature indicates that the portability of currently available quantitative software estimation models is poor. A primary reason is that most models fail to account for managerial characteristics of the software development environment; a set of factors that tend to vary significantly from one organization to another. A major stumbling block has been the inability to quantify the impact of managerial-type factors on cost.

In this study, we take a first step towards rectifying this situation. An extensive simulation model of the software development process is developed and used to identify managerial factors that impact the cost of software development, and to quantify the degree of that impact. Because the areas identified are variables that the project manager can objectively evaluate at the beginning of a software project, it should be feasible to incorporate them in future cost estimation models. This, we feel, would improve both their accuracy and portability.


Key Words: Software Estimation, Software Projects, Software Project Management, System Dynamics, Simulation.

## Introduction

Software development cost and schedule estimation continues to be a major difficulty in the management of software development [4]. While several quantitative software estimation models have been developed and widely publicized in the literature (e.g., [27] and [7]), still "...almost no model can estimate the true cost of software with any degree of accuracy" [24]. Furthermore, the portability of such models (i.e., in maintaining a good level of estimation accuracy when utilized in a different organization) has proven to be quite poor [6]. As a result, many software development organizations do not seem to trust any of the available quantitative models. A recent study of 30 organizations showed that the models were used only to "check manual estimates" [37].

> The significance of the problem was stated as follows:
> Unable to estimate accurately, the manager can know with certainty neither what resources to commit to an effort nor, in retrospect, how well these resources were used. The lack of a firm foundation for these two judgements can reduce programming management to a random process in that positive control is next to impossible. This situation often results in the budget overruns and schedule slippages that are all too common ... [16].

The thesis of this paper is that both the accuracy as well as the portability of software estimation models can be improved by taking into consideration not only the technical variables (e.g., source language, computer hardware characteristics, database size, ... etc.) as in most of the current models, but, in addition, explicitly to incorporate the managerial and organizational characteristics of the environment. Specifically, we identify a number of managerial variables that most current models fail to "acknowledge," but which significantly influence the cost of software development.

Mohanty's Experiment

We first report on an interesting experiment by Mohanty [24], which demonstrates the weaknesses in current models. His objective was to examine the extent to which 12 available quantitative software estimation models produce the same cost estimate for a given project. In order to specify his hypothetical software project, it was necessary to identify the full set of factors that are collectively incorporated in the 12 models. Forty-nine factors were identified. They involved system size, database size, system complexity, type of program, documentation, technical environment (e.g., requirements definition, security, and computer access), and an "other" category that includes such items as miles traveled, reliability, and growth requirements.

The hypothetical project was then defined in terms of the identified parameters. The project was chosen to be 36,000 machine-language executable instructions. The cost estimates for the project are exhibited in Figure 1. As this illustrates, the estimated cost varied from a low of $362,500 (the Farr and Zagorski Model) to a high of $2,766,667 (the Kustanowitz Model).

Two sources for the variations between the cost estimates were suggested. The first related to the quality of the final product. The second is environmental:

> ... That is, each model was developed for a cost data base collected in a given company environment. This data base thus embodies the specific nature of the organizational problems, work patterns, and management approaches and practices. Where this data base is regressed to derive coefficients for use in a given model, the model reflects that company's environment only [24].

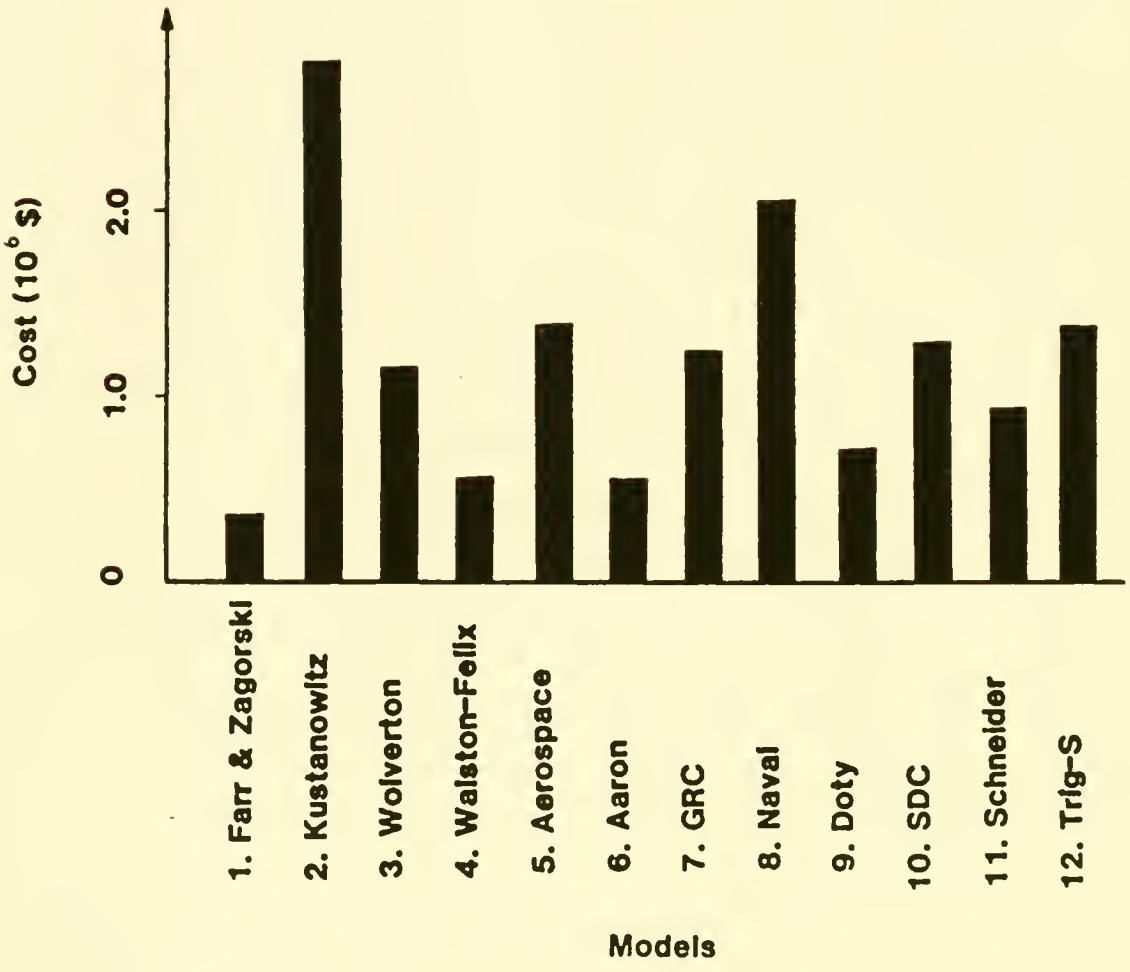


Figure 1

Cost estimates for a software project in Mohanty's Experiment

This view is supported by others in the literature ([12], [30], and [5]).

An Integrated System Dynamics Model of Software Development

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 [31]. In the words of Jensen and Tonies [20]: "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 --- a process that can and should be optimized."

The model we are presenting in this paper provides such an integrative perspective. It integrates, as will be demonstrated in more detail later, 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 important 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. The significance and applicabilty of the feedback systems concept to managerial systems has been substantiated by a large number of

studies [28]. For example, Weick [34] 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, ...

It is no wonder, then, that many software managers get into trouble because they forget to think in circles. We 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.

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.

Several authors (e.g., [31]) have complained about the lack of tested ideas in the software engineering field. Weiss [35] commented that in software engineering it is remarkably easy to propose hypotheses and remarkably difficult to test them. Accordingly, it is useful to seek methods for testing software engineering hypotheses.

Unfortunately, controlled experiments in the area of software development tend to be costly and time consuming [25]. Furthermore, even when it can be afforded the isolation of the effect and the evaluation of the impact of any given practice within a large, complex and dynamic project environment can be exceedingly difficult [18].

In addition to permitting less costly and less time-consuming experimentation, simulation models make "perfectly" controlled experimenation possible. Indeed:

> The effects of different assumptions and environmental factors can be tested. In the model system, unlike 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 [17].

Overview of Model Structure

The model was developed on the basis of a battery of 27 field interviews of software project managers in five software producing organizations, supplemented by extensive empirical findings from the literature. Figure 2 depicts the model's four subsystems, namely: (1) The Human Resource Management Subsystem; (2) The Software Production Subsystem; (3) The Control Subsystem; and (4) The Planning Subsystem. It also shows some of the interrelatedness of the four subsystems. The model integrates our knowledge of micro components (e.g., scheduling, programming, productivity) into a more continuous view of the software development process.

The Human Resource Management Subsystem captures the hiring, training, assimilation, and transfer of the human resource. Such actions are not carried out in vacuum, but are affected by the other subsystems; e.g., the

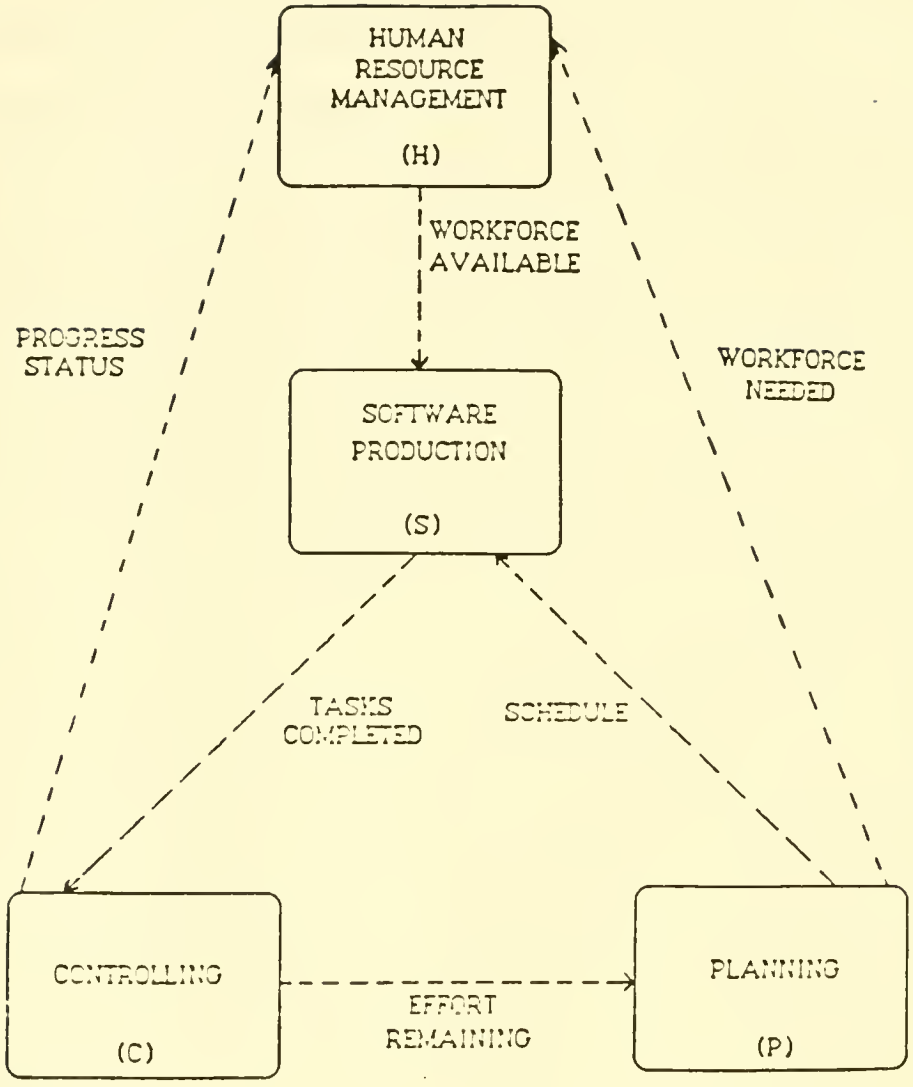


Figure 2

Four subsystems of model

hiring rate is a function of the workforce level needed to complete the project by a given date. Similarly available workforce has direct bearing on the allocation of manpower among the different production activities.

The four main software production activities are: development, quality assurance, rework, and testing. The development activity comprises both the design and coding of the software. As it is developed, it is also reviewed to detect any errors; e.g., with structured walkthroughs. Errors detected through such activities are then reworked. But not all errors are detected, some "escape" detection until the testing phase.

As progress is made, it is reported. A comparison of degree completed to planned schedule is captured within the Control Subsystem. Once an assessment of the project's status is made, it becomes an important input to the planning function.

In the Planning Subsystem, initial project estimates are made and then revised, when necessary, throughout the project's life. For example, to handle a project that is perceived to be behind schedule, plans can be revised to (among other things) hire more people, extend the schedule, or do both.

A full description of the model and of its validation are provided elsewhere ([1] and [3]). And in [2], we demonstrate the model's accuracy in reproducing the dynamic behavior of a real software project.

Model Experimentation:

In a series of experiments, we tested and quantified the impact of four-managerial type variables on the cost of software development. Two address manpower acquisition and staffing considerations, while the other two concern managerial judgement on the distribution of effort among the software development activities. The four variables were selected with two criteria in mind. The two criteria were proposed in [8]: objectivity and

prospectiveness. According to these authors, software cost estimation models should only include objective variables; this avoids allocating variance to poorly calibrated subjective factors. Thus it is harder to manipulate the model to obtain wanted results. Secondly, a model should avoid the use of variables that cannot be quantified until the project is complete.

Manpower-Acquisition and Staffing Variables:

The two model variables that address manpower-acquisition policy issues are: (1) fractional time on job; and (2) the willingness to change workforce level.

Field interviews revealed differences in project staffing policies. In some organizations project members were assigned full-time to a single project, whereas at others, software developers were assigned to more than one (usually two) [21]. In the model, this issue is captured by the variable "Fractional Time on Job." For example, when project members are assigned full-time to a project, the value of the "Fractional Time on Job" would be set to 1 i.e., each project member contributes 1 man-day every (working) day to the project. On the other hand, if project members allocate, on the average, only 50% of their time to the project, the value of the "Fractional Time on Job" would be set to 0.5.

To examine the impact of these two different staffing policies on project cost, we defined the EXAMPLE software project for the simulation experiment, and then conducted two simulation runs. (In the Appendix, a parameter profile of the EXAMPLE software project is presented together with its base case simulation run.) In the first, the value of the "Fractional Time on Job" was set to 1, and in the second it was set to 0.5. The measure of the project cost we will use is the value of the total number of man-days expended to complete the project. The results were as

follows:

| Fractional Time on Job | Man-Days |
|---|---|
| 1.0 | 3,795 |
| 0.5 | 4,641 |

In other words, the policy of allocating project members half-time to the project results in a 22% higher cost. The reason for this increase is two-fold. First, there is a loss in productivity due to the increase in the communication overhead. This factor accounts for approximately 90% of the increase. The average staffing level of a project (in terms of full-time equivalent employees) is typically determined by dividing the estimated size of the project in man-days by the project's estimated development time [7]. When the "Fractional Time on Job" is less than 1, an upward adjustment is obvoiusly needed. For example, if a project's size is 1000 man-days and its scheduled duration is 200 days, the average staffing level for full-time equivalent employees would be 5. But if employees are assigned only half-time, then the staffing level would be raised to 10. This increases the time lost on human communication, e.g., to resolve questions [30]. In addition, the amount of project work itself usually increases; e.g., in the form of more documentation, more modules, and more interfaces [13]. The result is a decrease in productivity.

The second reason why the cost increases is because of an increase in the training overhead. This factor accounts for the remaining 10% of the increase. When new project members are recruited (from within the organization or from the outside), they often pass through a project orientation period. This training of newcomers is usually carried out by the "old timers" ([29] and [36]). This overhead is, of course, costly, because while (the oldtimer) is helping the new employee learn the job, his own productivity on his other work is reduced [11]. This training overhead is a function of the number of newcomers, not of the number of equivalent full-time newcomers. In [19], when project members were

assigned half-time on the project, the team size was doubled, which indeed doubled the training overhead incurred on the project.

The second manpower-acquisition variable tested is the "Willingness to Change Workforce." When deciding upon changes in the workforce level (because the project is falling behind schedule) software project managers typically consider a number of factors. One is the scheduled completion date. As part of the continuous planning function management determines the workforce level necessary to complete the project within the scheduled time. In addition, consideration is given to the stability of the workforce. Thus, before hiring new project members, management tries to decide how long it will utilize the new members. Different firms weigh this factor to various extents. In general, however, the relative weights change with the stage of the project. Toward the end of the project there is usually considerable reluctance to bring in new people, even though the time and effort remaining might imply that more are needed. It would take too much of the remaining project time to acquaint new people with the mechanics of the project, integrate them into the team, and train them in the necessary technical areas.

These managerial considerations are entered into the model with a weight factor termed "Willingness to Change WorkForce" (WCWF). It is a variable that could assume values from 0 to 1. When WCWF = 1, the "Workforce Level Sought" would simply be set equal to the "Workforce Level Perceived Needed;" i.e., management would be adjusting its workforce level to finish on schedule, determined by dividing the amount of effort that management perceives is still remaining by the time remaining to complete the project. As WCWF moves towards 0, more and more weight would be given to the stability of the workforce. And when WCWF is 0, the "Workforce Level Sought" becomes equal to the "Current Workforce" i.e., management attempts to maintain the workforce at its current level. A WCWF value

between 0 and 1, on the other hand, represents a situation where management responds to schedule slippages by partially increasing the workforce level (workforce level sought would be set to a value less than, not equal to the workforce level perceived needed to complete the project on the current schedule) and partially extending the current schedule to a new date. Thus,

Workforce Level Sought = (Workforce Level Perceived Needed)*(WCWF) +
(Current Workforce)*(1-WCWF)

Note: The above formulation only applies when the value of the "Workforce Level Perceived Needed" is larger than "Current Workforce," indicating a need for hiring more people. In cases where this is not true, the "Workforce Level Sought" would be set to the lower value, and any excessive employees transferred out of the project.

It is important to realize that the variable "Willingness to Change Workforce" (WCWF) is an expression of a policy for managing projects. For any specific project environment, the WCWF function can be derived on the basis of interviews with project managers as well reviews of historical project records.

The "Willingness to Change Workforce" (WCWF) curve depicted in Figure 3 characterizes the workforce acquisition policy in one organization studied, a large American minicomputer manufacturer. In the early stages of the project when "Time Remaining" would generally be much larger than the sum of the hiring and assimilation delays (the latter being the time needed for a new team member to become fully productive), WCWF is one; i.e., there is total willingness to adjust the size of the workforce to whatever level is necessary to suit the scheduled completion date. This inclination to respond to any schedule slippages in the early phases of the project through adjustments in the workforce level rather than adjustments in the schedule is mainly a result of political

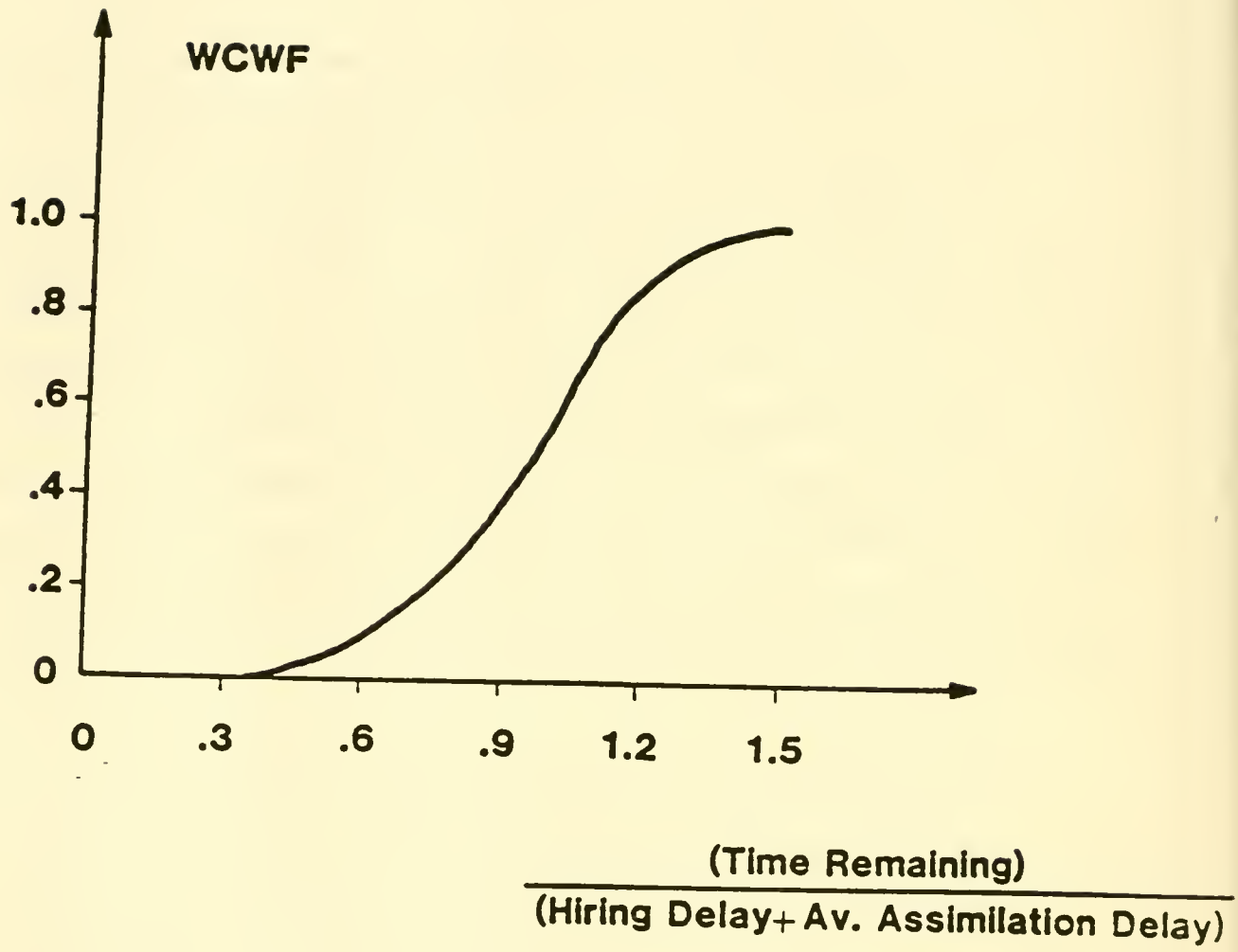


Figure 3

Willingness to Change Workforce (base case)

pressures:

> Once an original estimate is made, it's all too tempting to pass up subsequent opportunities to estimate by simply sticking with your previous numbers. This often happens even when you know your old estimates are substantially off. There are a few different possible explanations for this effect: 'It's too early to show slip' ... 'If I re-estimate now, I risk having to do it again later (and looking bad twice)' [14].

As the number of days perceived remaining drops below 1.5*(Hiring Delay + Assimilation Delay), though, there is reluctance to increase the workforce level. For example, if the "Hiring Delay" is 40 working days (i.e., eight calendar weeks) and the "Assimilation Delay" is 80 working days, then as "Time Remaining" drops below 180 working days, management becomes reluctant to add new people, even though the effort remaining might imply (at that point in the project) that more people are needed. This reluctance, as mentioned above, stems from the realization that most of these remaining days would be "wasted." When the "Time Remaining" drops below 0.3*(Hiring Dalay + Assimilation Delay), the particular policy curve of Figure 3 suggests that no more additions would be made to the project's workforce. Thus, if the project is behind schedule, project management would cope with project slippage through adjustment to the completion date. We term this policy the base case.

One other manpower acquisition policy that we shall term policy (A), can be defined as follows: At the initiation of the project, estimates are made of the total effort in man-days (MD) and development time (TDEV). Based on these, the staffing level is determined; i.e., by dividing MD by TDEV. People are hired, usually to complement the core project team on hand at the initiation of the project, until the desired staffing level is

reached. Then, the workforce is maintained at that level, with new people hired only to replace those who quit or are transferred. Such a policy was also reported by Devenny [15] in his study of software cost estimation.

The simulation results of this policy, together with that of the base run, are:

| Manpower Acquisition Policy | Man-Days | Duration (Days) |
|---|---|---|
| Base Case | 3,795 | 430 |
| A | 3,559 | 488 |

As the figures indicate, policy (A) leads to a 6% drop in cost. However, this is achieved at the cost of a larger schedule slip, because the project takes 13.5% more time to complete. Whether this tradeoff is made consciously is not clear. However, by foregoing the flexibility of adjusting the workforce level, this staffing policy leaves little room in handling any project delays beyond translating them into completion slippages.

Under a third manpower acquisition policy we examined, policy (B), project management is not only willing to adjust the workforce level (e.g., to account for any initial underestimation error), but is willing to continue making such adjustments further into the project life cycle (that is, further than in the base case). This policy is adopted by one growing software consulting company. The WCWF curve for policy (B) is shown in Figure 4. The major difference between this and the base policy of Figure 3 is that the denominator of the X-axis variable is simply the "Hiring Delay" rather than the sum of the "Hiring Delay" and the "Assimilation Delay." This, of course, means that policy (B) is a more aggressive policy in terms of acquiring people. While in the base case policy management is reluctant to increase the workforce level when the perceived number of days remaining to complete the project drops below 1.5 *(Hiring Delay + Assimilation Delay), under policy (B) this happens much

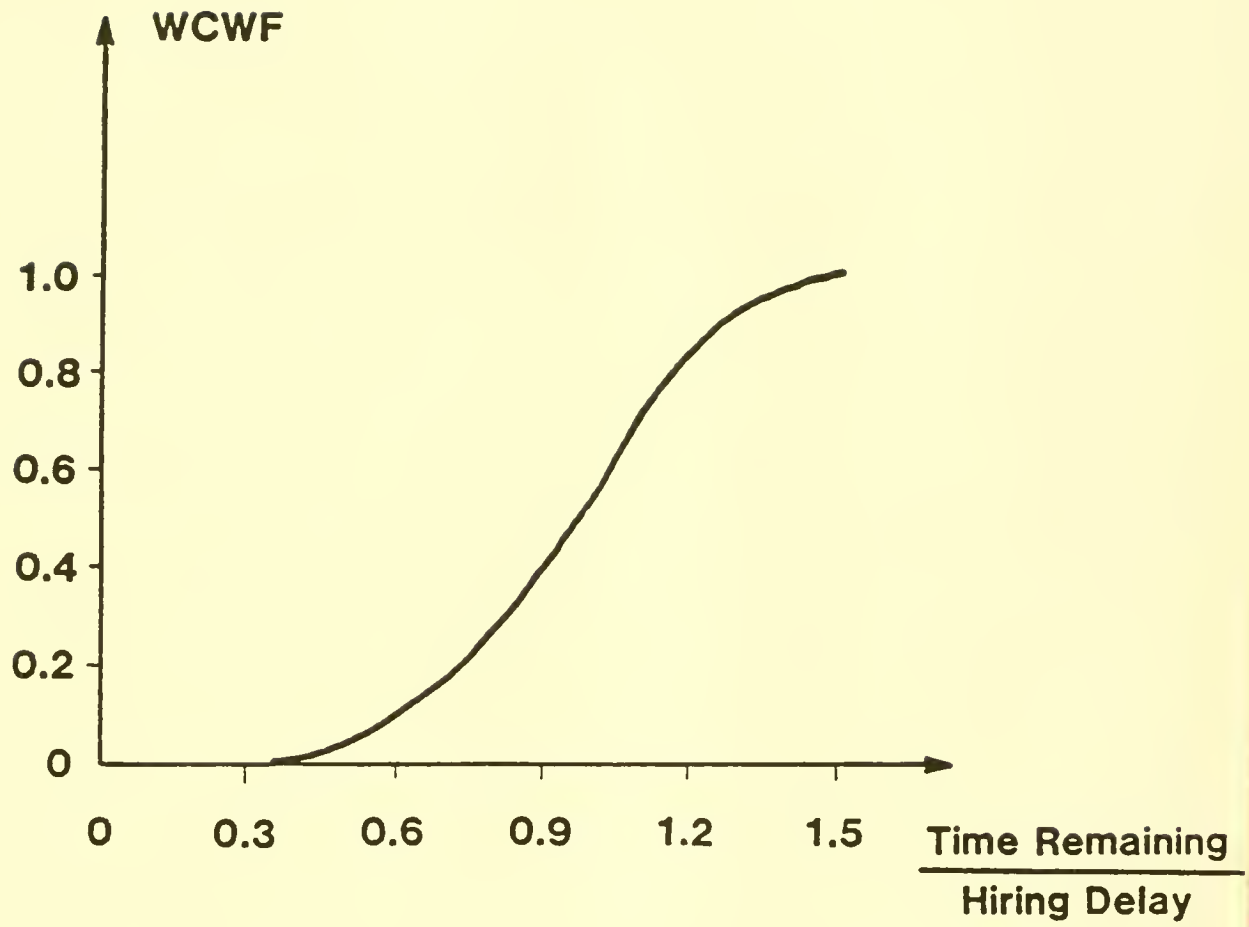


Figure 4

Willingness to Change Workforce (Policy B)

later in the project's lifecycle (i.e., when only 1.5*40=60 working days are perceived remaining). This policy is justified, we were told, because the firm is experiencing an impressive growth rate, fueled by a sizable backlog of client assignments. Hiring new people into a project that is "winding down" is, therefore, not inhibited by management since securing the future utilization of the new people is almost always guaranteed.

The result of adopting such a policy in project EXAMPLE is shown below, together (for the reader's convenience) with the results of both the base case and policy (A).

| Manpower Acquisition Policy | Man-Days | Duration (Days) |
|---|---|---|
| Base-Case | 3,795 | 430 |
| A | 3,559 | 488 |
| B | 4,322 | 373 |

As the figures indicate, cost in using policy (B) is 14% higher than the base case and 21% higher than that of policy (A). On the other hand, under policy (B) the project takes 13% less time to complete than the base case, and almost 25% less time than when policy (A) is used. Both the increase in the cost and the decrease in the duration can be attributed to a single cause, namely, a higher workforce level that results from the increased willingness to add people to the project.

Effort Distribution Variables:

In planning a software project, management not only provides estimates for the project's total expenditure, it also plans the distribution of this effort among the project's phases. Numerous authors have presented figures indicating lifecycle resource distributions. A comparison of several by McKeen [22], indicated that substantial differences do exist, particularly in the coding and testing phases. Commenting on the situation, he says: "A major conclusion ... is that we do not possess an

adequate understanding of resource consumption behavior over the life cycle development phases." He studied 32 software development projects, and found no real support for typical or dominant development profiles at all.

We now examine the impact of the distribution of effort among the project's phases in our prototype on project cost. The model has two effort distribution parameters. The first concerns the allocation of the project's estimated man-days among the model's two major phases: development (design and coding) and system testing. In the base case, 80% of the effort is allocated to development and 20% to testing. The second parameter is the "Planned Fraction of Manpower for Quality Assurance." In the base case this is set to 15% i.e., 15% of the development effort is allocated (in the project's plan) for QA activities during the design and coding stages.

For our simulation, we selected a second effort distribution profile that was used in a major auto manufacturer, the 40-20-40 effort distribution profile i.e., 40% for preliminary and detailed design, 20% for coding, and 40% for testing. We should note that this 40-20-40 rule is perhaps the most widely touted rule-of-thumb on the distribution of effort ([20], [10], and [26]). As for QA, the particular organization's software project teams allocated, on average, 20% of the development effort to quality assurance.

The result of running project EXAMPLE with this new effort distribution profile, termed (C), were:

| Effort Distribution Profile | Man-Days |
|---|---|
| Base Case | 3,795 |
| C | 4,443 |

Thus, a change in the effort distribution profile from the base case to profile (C) leads to a 17% increase in cost. Several factors contributed to this. The most significant factor is the increase in the cost of the development phase.

Consider the sequence of steps in planning a project's various activities (e.g., [7]). First, total man-days is estimated. Based on this, the schedule is calculated. The two estimes are then used to determine the average staff size. And allocations are then made to the project's various lifecycle activities. Effort distribution decisions typically come after, not before, the project's schedule is made. Thus if two different project managers were to run the same software project and if the only thing that differentiates the two is their policies on distributing the effort, both would still initiate their project with the same global estimates. It is exactly this type of scenario that we attempted to capture in our simulation. Thus, in both runs the project's total man-day estimate as well as the scheduled completion date remain the same. But since in profile C a lower fraction of the manpower is devoted to development work (because of the increased allocation to QA), a larger team will be required to meet the schedule. A larger team means larger training and communication overheads, and hence the larger development cost.

A Final Experiment

In the final experiment, we examined the combined effect of the four variables on project cost. This is achieved by the following four adjustments:

1. Setting the value of the "Fractional Time on Job" to 0.5. (The base case value is 1.)

2. The "Willingness to Change Workforce" is formulated in terms of the "Hiring Delay," yielding a more aggressive manpower acquisition policy. [In the base case it is formulated in terms of the (Hiring Delay + Assimilation Delay).]

3. Allocation of effort among the development and testing phases is set at 60% development (design and coding) and 40% testing. (In the base case it is 80-20.)

4. The "Planned Fraction of Manpower for QA" is set at 20%. (In the base case 15%.)

The result of this different set of managerial policies is a total cost of 7,316 man-days. That is, a cost that is almost double the base case cost of 3,795 man-days.

Conclusion

The implication of the results is clear: because managerial policies vary from software development organization to another and because they impact the cost of software development, the portability of software cost estimation models would be improved if such variables are explicitly incorporated in the models' formulations.

Heretofore, the impact that a company's managerial environment can have on its software development costs has not been quantified. We feel that our work produced three useful results. First, we have established that the impact is significant; specifically, we have shown that the combined effect of four managerial variables can increase the cost of a software project by at least a factor of 2. Secondly, by quantifying the

individual variables' impacts, we have taken a first step towards the incorporation of such mangerial variables in the formulation of software estimation models. Such an enhancement would undoubtedly improve the portability of the models. Finally, we have identified four aspects of the managerial environment of software development that are significant determinants of software development cost, and which are, therefore, deserving of further research.

# APPENDIX

## Software Project EXAMPLE

| | | |
|---|---|---|
| 1. Project Size | = 64,000 | DSI |
| 2. Man-Days | | |
| Total | = 3,795 | Man-days |
| Development | = 2,681 | " |
| Design & Coding | = 1,782 | " |
| Quality Assurance | = 380 | " |
| Error Rework | = 519 | " |
| Testing | = 1,114 | " |
| 3. Completion time | = 430 | Working days |

Project EXAMPLE's base case simulation run is depicted in Figure 5 below.

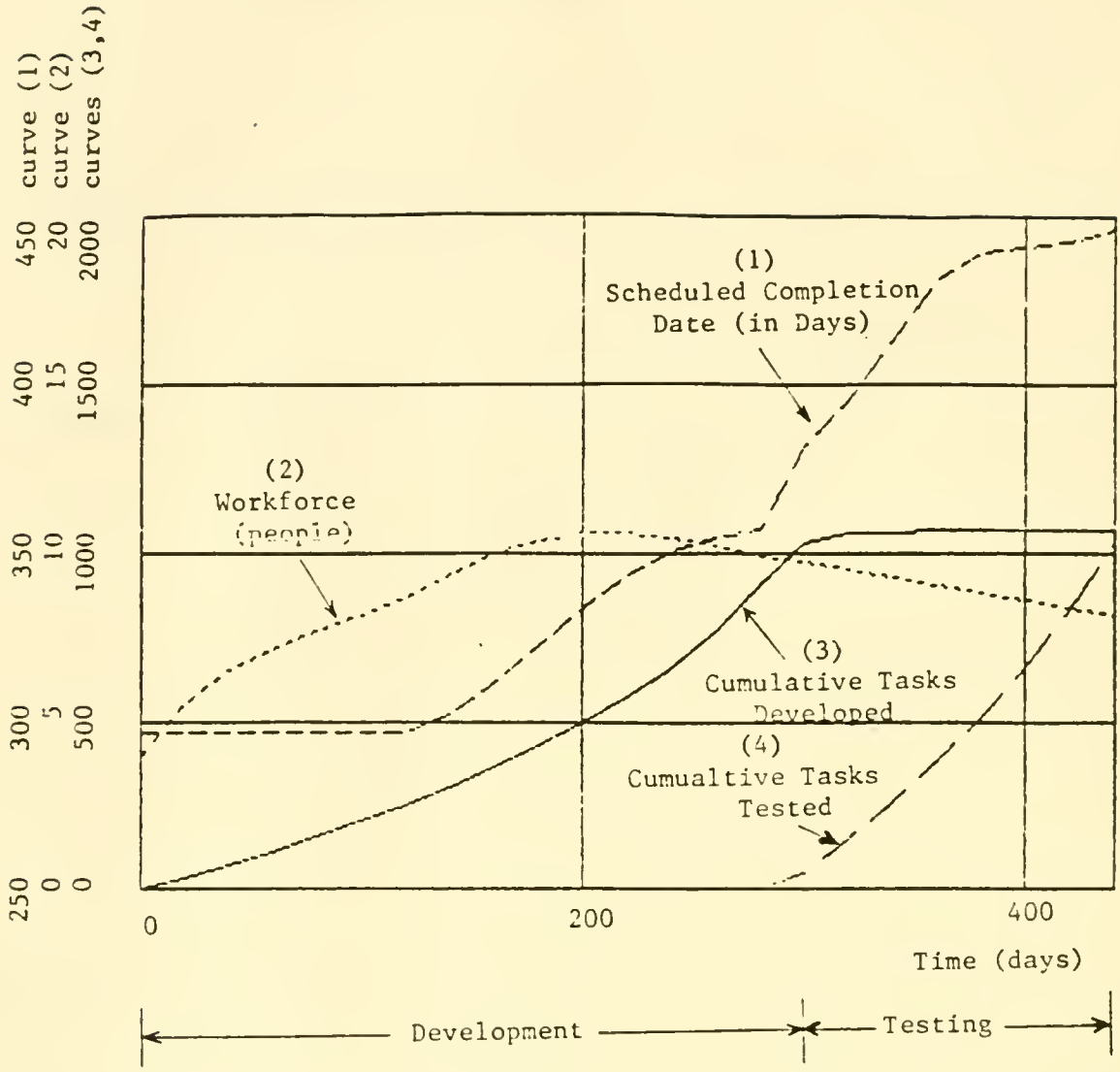


Figure 5

Project EXAMPLE's base case simulation run

Bibliography

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

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

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

[4] M.R. Barbacci, A.N. Habermann, and M. Shaw. "The Software Engineering Institute: Bridging Practice and Potential." IEEE Software, November 1985, 4-21.

[5] K.M. Bartol and D.C. Martin. "Managing Information Systems Personnel: A Review of the literature and Manaherial Implications." MIS Quarterly, Dec. 1982, 49-70.

[6] I. Benbasat and I. Vessey. "Programmer and Analyst Time/Cost Estimation." MIS Quarterly, June, 1980, 31-44.

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

[8] B.W. Boehm and R.W. Wolverton. "Software Cost Modeling: Some Lessons Learned." J. of Systems and Software, Vol 1, No. 3, 1980.

[9] F.P. Brooks. The Mythical Man Month. Reading, Mass: Addison-Wesley Publishing Co., 1975.

[10] P. Bruce and S.M. Pederson. The Software Development Project: Planning and Management. New York: John Wiley & Sons, Inc., 1982.

[11] R.G. Canning. "Managing Staff retention and Turnover." EDP Analyzer, Aug., 1977, 1-13.

[12] J.A. Clapp. "A Review of Software Cost Estimation Methods." MITRE Technical Report, June 1976, 1-55.

[13] M.E. Conway. "How do Committees Invent." Datamation, April, 1968, 28-31.

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

[15] T.J. Devenny. "An Exploration Study of Software Cost Estimating at the Electronic Systems Division." NTIS, U.S. Dept. of Commerce, July, 1976.

[16] J.A. Farquhar. A Preliminary Inquiry into the Software Estimation Process. Technical Report, AD F12 052, Defence Documentation Center, Alexandria, Va., Aug., 1970.

[17] J.W. Forrester. Industrial Dynamics. Cambridge, Mass: The MIT Press, 1961.

[18] R.L. Glass. Modern Programming Practices: A Report from Industry. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1982.

[19] R.L. Gordon and J.C. Lamb. "A Close look at Brooks' Law." Datamation. June, 1977, 81-86.

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

[21] J. Knutson. "Developing the Project Plan." In Advances in Computer Programming Management. Edited by T.A. Rullo. Philadelphia: Heyden & Sons, Inc., 1980.

[22] J.D. McKeen. "An Imperical Investigation of the process and Product of Application System Development." Unpublished Ph.D. dissertation, University of Minnesota, 1981.

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

[24] S.N. Mohanty. "Software Cost Estimation: Present amd Future." Software---Practice and Experience, 1981, 103-121.

[25] G.J. Myers. "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections." Comm. of the ACM, Sept., 1978, 760-768.

[26] P. Oliver. "Estimating the Cost of Software." In Computer Programming Management. Edited by J. Hannan. Pennsauken, New Jersey: Auerbach Publishers, Inc., 1982.

[27] L.H. Putnam. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem." IEEE Tr. on Software Engineering, July, 1978.

[28] E.B. Roberts (ed.). Managerial Applications of System Dynamics. Cambridge, Massachusetts: The MIT Press, 1981.

[29] M.R. Tanniru et al. "Causes of Turnover among DP professionals." Proc. of the 8th Annual Computer Personnel Research Conf., Miami, Florida, June, 1981.

[30] R.C. Tausworthe. Standardized Development of Computer Science. Englewood cliffs, N.J.: Prentice-Hall, Inc., 1977.

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

[32] R.H. Thayer and J.H. Lehman. Software Engineering Project Management: A Survey Concerning U.S. Aerospace Industry Management of Software Development Projects. Sacramento Air Logistics Center, McClellan Air Force Base, California, November, 1977.

[33] R. Thibodeau and E.N. Dodson. "Life Cycle Phase Interrelationships." Journal of Systems and Software, Vol. 1, 1980, 203-211.

[34] K.E. Weick. The Social Psychology of Organization. Second Edition. Reading, Massachusetts: Addison-Wesley Publishing Co., Inc., 1979.

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

[36] Winrow. "Acquiring Entry-Level Programmers." In Computer Programming Management. Edited by J. Hannan. Pennsauken, New Jersey: Auerbach Publishers, Inc., 1982.

[37] M.V. Zelkowitz et al. "Software Engineering Practices in the US and Japan." Computer, June, 1984, 57-66.