Contract Number N00039-78-G-0160 (Order 002)

Internal Report Number P010-7905-10

Deliverable Number 003

A NORMATIVE COST-BENEFIT ANALYSIS OF THE SYSTEMATIC DESIGN METHODOLOGY

Technical Report #10

S.L. Huff

May 1979

Principal Investigator:

Prof. S.E. Madnick

Prepared for:

Naval Electronic Systems Command
Washington, D.C.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report #10	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Normative Cost-Benefit Analysis of the Systematic Design Methodology		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER P010-7905-10
7. AUTHOR(s) S. L. Huff		8. CONTRACT OR GRANT NUMBER(s) N00039-78-G-0160
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center For Information Systems Research Sloan School of Management, MIT Cambridge, Mass. 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Electronic Systems Command		12. REPORT DATE May 1979
		13. NUMBER OF PAGES 65
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software architectural design; problem design structuring; normative cost-benefit analysis

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Complex design problems are characterized by a multitude of competing requirements. System designers frequently find the scope of the problem beyond their conceptual abilities, and attempt to cope with this difficulty by decomposing the original design problem into smaller, more manageable sub-problems. Functional requirements form a key interface between the users of a system and its designers. In this research effort, a systematic approach has been proposed for the decomposition of the overall set of functional

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

requirements into sub-problems to form a design structure that will exhibit the key characteristics of good design: strong coupling within sub-problems, and weak coupling between them.

Recent work in the Systematic Design Methodology project has led to certain extenstion to the basic representational model used therein. This report presents a normative cost-benefit analysis of the SDM. The Systematic Design Methodology is a decision support methodology for aiding a software designer in determining an optimal structuring of a system's functional requirements. A model-oriented, normative cost/benefit analysis of the SDM is presented here. A set of three sub-models, pertaining to specification impact, procedural design impact, and maintenance/modification impact, are derived. These models attempt to capture, in functional form, the important quantifiable effects - both positive and megative - that ought to occur as a result of developing a system based o an optimal partitioning of requirements.

A numerical example is also discussed to better illustrate the kinds of results predicted by themodels, in a realistic design scenario.

PREFACE

The Center for Information Systems Research (CISR) is a research center of the M.I.T. Sloan School of Management. It consists of a group of management information systems specialists, including faculty members, full-time research staff, and student research assistants. The Center's general research thrust is to devise better means for designing, implementing, and maintaining application software, information systems, and decision support systems.

Within the context of the research effort sponsored by the Naval Electronics Systems Command under contract N00039-78-G-0160, CISR has proposed to conduct basic research on a systematic approach to the early phases of complex systems design. The main goal of this work is the development of a well-defined methodology to fill the gap between system requirements specification and detailed system design.

The research being performed under this contract builds directly upon results stemming from previous research carried out under contract N00039-77-C-0255. The main results of that work include a basic scheme for modelling a set of design problem requirements, techniques for decomposing the requirements set to form a design structure, and guidelines for using the methodology developed from experience gained in testing it on a specific, realistic design problem.

The present study aims to extend and enhance the previous work, primarily through efforts in the following areas:

1) additional testing of both the basic methodology, and proposed extensions, through application to other realistic design problems;

2) investigation of alternative methods for effectively coupling this methodology together with the preceding and following activities in the systems analysis and design cycle;

3) extensions of the earlier representational scheme to allow modelling of additional design-relevant information;

4) development of appropriate graph decomposition techniques and software support tools for testing out the proposed extensions.

In this report, a cost-benefit analysis of the new techniques proposed in the SDM is carried out. Three models are developed which attempt to capture, in functional form, the important quantifiable effects that ought to occur as a result of developing a system based on an optimal partitioning of requirements. Some numerical examples are carried out to illustrate the kind of results predicted by the models in a realistic design scenario.

EXECUTIVE SUMMARY

Complex design problems are characterized by a multiple of competing requirements. System designers frequently find the scope of the problem beyond their conceptual abilities, and attempt to cope with this difficulty by decomposing the original design problem into smaller, more manageable sub-problems. Functional requirements form a key interface between the users of a system and its designers. In this research effort, a systematic approach has been proposed for the decomposition of the overall set of functional requirements into sub-problems to form a design structure that will exhibit the key characteristics of good design: strong coupling within sub-problems, and weal coupling between them.

Recent work in the Systematic Design Methodology project has led to certain extensions to the basic representational model used therein. This report presents a model-oriented normative cost-benefit of the Systematic Design Methodology. A set of three sub-models, pertaining to specification impact, procedural design impact, and maintenance/modification impact, are derived. These models attempt to capture, in functional form, the important quantifiable effects - both positive and negative - that ought to occur as a result of developing a system based on an optimal partitioning of requirements.

A numerical example is also discussed to better illustrate the kinds of results predicted by the models, in a realistic design scenario.

TABLE OF CONTENTS

1. Introduction.

The Systematic Design Methodology (SDM) is a decision support-oriented methodology for aiding a software designer in determining a good design architecture, or preliminary structuring, of the functional requirements for the system under design. Various aspects of the methodology itself have been described elsewhere ((Andreu 78), (Huff and Madnick 78a,b), (Huff 79)).

In this report, an analysis of the economic impact of SDM is presented. This analysis is embodied in a normative, conceptual cost/benefit model (actually, a set of three "sub-models"). The model attempts to represent, in functional form, the important quantifiable results - both positive and negative - that would be extected to occur as a result of adopting and using SDM in software design projects.

Three major categories of beneficial impact are examined:

a) system improvements resulting from more accurate and appropriate requirements definition;

b) reduced time and cost for detailed design and implementation ("procedural design") resulting from lower communication and control overhead achievable through an improved system partitioning;

c) reduced cost for making later modifications to the final system, resulting from reduced inter-module "ripple effect."

The major category of detremental impact addressed here is that of costs (essentially staff time) related to the carrying out of the SDM activities that would not otherwise have to be carried out.

While there are potentially many other categories of costs and benefits that might be attributable to the use of a design methodology such as SDM, it is believed that those identified above effectively capture the first-order impact. Other potential impacts are either so intangible as to be unmodellable quantitatively (e.g., the psychological impact of the methodology upon the system designers), or are of a marginal magnitude relative to the above categories (e.g., the impact of SDM on the code debugging process).

Calculations for a hypothetical but realistic example are carried out to illustrate the nature of the model further.

1.1 Background and Related Work.

A number of authors have stressed the shifting nature of the economics of computer-based systems (e.g., Dolotta, et. al. 76). While the cost of computer hardware, notably logic and memory, continues to fall at a rate exceeding 20% per year, costs of software development are moving in the opposite direction, nearly as fast (Wasserman, et. al. 78). The dramatic decrease in hardware costs is related to the steep learning curve for micro-circuit fabrication

techniques, and to the economics of mass production. The very first Intel 8080 microprocessor "cost" about $400 million, but the second and subsequent copies only cost about $0.27!

In contrast, productivity improvements in the systems design and programming fields have come much more slowly. The central objective of the ten-year-old software engineering field is to develop new ways of "manufacturing" computer software systems that will make possible productivity gains comparable to those being achieved in the hardware side of the industry. Nonetheless, Boehm's analysis of software production for the Air Force (Boehm 73) indicates that typical software costs for major systems may approach 90% of the total system (hardware plus software) cost in the near future, as shown in Figure 1.1.

While progress in software engineering has been considerably less dramatic than in the hardware area, some important advances have occurred. There is also a substantial body of reserach presently under way. An important outgrowth of these facts, not yet widely recognised, is the need for software "research on research": for the development of methodologies, models, techniques, etc. for evaluating the successfulness of the various new approaches to software design and development. It is this need that the present study addresses.

The literature pertaining directly to evaluation of software design/development is almost nonexistant. However,

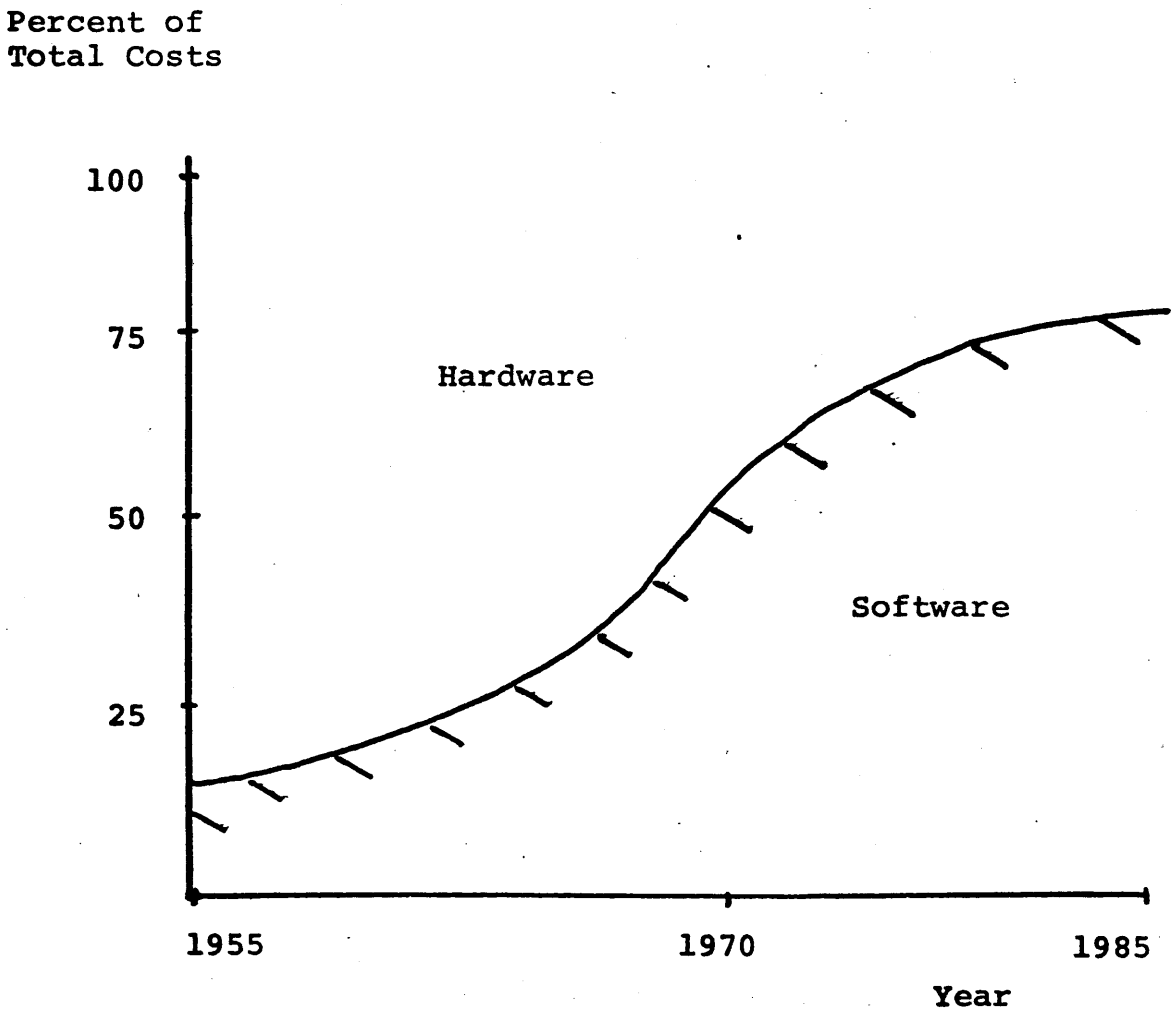

Figure 1.1

source: (Boehm 73)

some studies have been reported in related areas. Probably the most frequent target of attention has been productivity levels within the programming and system development activities. Key studies include those of Aron (Aron 70), Wolverton (Wolverton 74), Putnam (Putnam 78), and Walston and Felix (Walston and Felix 77). These studies have basically addressed two main concerns:

a) what is programmer (developer) productivity, and how ought it be measured?

b) what factors influence productivity, and how?

Walston and Felix, for example, have identified and ranked 29 different influencing factors, using as a productivity index the ratio delivered source lines of code to total effort in man-months.

Another set of studies have addressed the system development cycle. Many of these studies present normative frameworks, extracted from the authors' experience; for example, (Cooper 78), and (Cave and Salisbury 78). Others have developed models of the system development process, and use them to try to better understand the effects of various parameters on the process, and to serve as the basis for process planning and control methodologies (e.g., Putnam 78).

A few other studies address specific stages of the system development cycle - for example, software testing (Bate and Ligler 78), or system maintenance (Lientz,

et. al. 77). Most of these studies also attempt to identify the important independent and dependent variables of the associated process, to develop models relating the sets of variables, and to analyze the models so as to gain further insight into the nature of the process.

1.2 Evaluating Design Methodologies.

As stated earlier, there is a conspictuous void in this literature, namely, studies of the economic and other impacts that system design procedures have on the development cycle, as well as upon the containing organization. A number of new ideas about how to design software effectively, including development of new conceptual frameworks, methodologies, and tools for design, have recently appeared. Examples of these include:

- Structured Design (Stevens et. al. 74)
- HIPO (Stay 76)
- SREM (Davis & Vick 77)
- PSL/PSA (Teichroew & Bastarache 75)
- SADT (Ross & Schomann 77).

Most of these schemes are quite new, however, and there has been almost no effectiveness, or cost/benefit, evaluation studies reported for them. Reasons for this include:

- the newness of the methodologies themselves - there simply hasn't been time yet to undertake serious evaluation;

- the problem of what to evaluate - i.e., measurement of software design
- the question of how to evaluate - whether via models, empirical testing, user attitude surveys, or other means.

Particular questions depend upon the scheme being evaluated. However, speaking generally, there are two main approaches that may be taken to design methodology evaluation: normative, and descriptive. The normative approach may be described as follows. In theory, a design methodology X should result in certain kinds of design and related improvements, and incur certain costs. The normative evaluation task involves developing theoretical models of the benefit and cost impacts of the methodology, then to explore the methodology's potential impact on the system life cycle economics using the models.

The descriptive approach, in contrast, is concerned with observing and studing the actual use of methodology X. In other words, a real (or possibly experimental) system design group using the methodology would be monitored by the evaluator, with the objective of isolating practical benefits and costs. Much of the gathered data would presumably be subjective, based on opinions and judgments of either the designers, the evaluator, or both. Evaluation analysis would also be subjective ("it seemed to be easier ..."), but might also involve standard statistical testing techniques.

A major difficulty with descriptive evaluation would be the fact that such studies should be longitudinal over the entire system life cycle (typically, ten years or more), as an important potential benefit of most design methodologies should be easier or better maintenance/modification activities. This effect would not be observable during the main development phase. Consequently the time to conduct a complete study would be rather extensive, and the costs quite high. This, of course, is yet another reason for the absence of such studies.

One other difficult issue regarding evaluation regards the appropriate base case. For instance, ought design methodology X be evaluated against the use of no formal methodology, or against the use of some other methodology Y. In the best possible world, multiple pairwise comparisons would be carried out, between X and each other appropriate "competitor" methodology. This approach is, clearly, far too expensive and resource-consuming for all but the wealthiest of organizations (e.g., the U. S. Department of Defense) to pursue.

1.3 Approach to be Taken.

The present study, being exploratory, takes the normative approach, and addresses the "null" base case. That is, the objective of this report is to identify and model the key benefit and cost issues that ought to arise as a result of using the Systematic Design Methodology.

Evaluations are made against the "no methodology" case as a base.

The key arguments and results of this analysis are presented in the next three sections. Each section addresses a different aspect of system design and development that the Systematic Design Methodology impacts. In each case, verbal arguments describing the nature of, and resources underlying, the impact category are given. Then the key effects in each case are represented in a functional model. This gives rise to three different sub-models. Section 5 is devoted to "pulling together" the sub-models, and addresses the ways in which they interrelate. Also, a brief hypothetical example illustrating an application of the models is given there. Section 6 contains concluding comments.

2. A System Specification Impact Model.

A common refrain from system developers, when asked why their systems are late, over budget, or even complete failures, has been that they were unable to secure adequate user participation in the development process. In fact, a survey of over 100 implementation success factor studies by Ginsberg found "user participation" to be the single universal factor (Ginsberg 74).

While there are many reasons why user participation is important in system development, perhaps the most important reason concerns system requirements correctness and completeness. The manifold difficulty of eliciting a user's complete set of requirements for a target system has been commented upon and studied by a number of researchers (e.g., Bell and Thayer 76). Others, such as Boehm, have argued strongly that not enough time is spent during the initial requirements elicitation and verification phase of the development process. Boehm calculates that there might be as much as a threefold return to extra effort expended during the early requirements analysis activities (Boehm 74).

One of the important impacts of SDM concerns requirements definition. SDM is "driven" by the set of system functional requirements; they play an even more central and crucial role in the SDM than in the usual system

development process. This comes about for three different reasons:

a) the need to develop requirements in statement form, which meet the SDM criteria (unifunctionality, implementation independence, common level of abstraction; see (Huff and Madnick 78a));

b) performance of interdependency analysis;

c) interpretation of a partitioning of requirements as a system architecture (Andreu 78).

Each of these are key activities within the Systematic Design Methodology. In carrying out each activity, the attention of the user and system developer are jointly focussed, in a relatively rigorous, structured way, upon the user-level (non-procedural) functional specifications of the target system. It is through this "forced," structured focussing of attention brought about by the need to carry out the steps of the methodology that missing requirements are identified, requirement inconsistencies spotted and resolved, and requirement statement errors discovered and corrected.

There is nothing magical about why this should occur. In essence, any approach that necessitates a careful, step-by-step analysis of requirements in such a structured fashion ought to achieve many of these same results. Within the SDM specifically, however, the interdependency analysis phase, and the partitioning interpretation phase, both direct extra light upon the kinds of problems that are

frequently observed to occur in this work. Interdependency analysis requires the designer (and user, to a lesser extent) to examine requirement reinforcements and tradeoffs on a pairwise basis, thereby bringing to light requirement inconsistencies and errors that might be overlooked if attention were not directed to such a specific detail level. Architectural design generation involves a general "reasonableness" assessment of the various groupings of requirements produced by the SDM graph decomposition, and experience has shown that it also helps to highlight missing requirements and requirement errors (Andreu 78).

Prior to presenting the impact model for requirements specification, certain underlying assumptions should be discussed.

Assumption 1. The requirements analysis and assessment activity transpires in a series of identifiable "passes." This assumption is borne out empirically, and also through a consideration of the SDM operational mechanisms. For example, a typical SDM-oriented development effort might follow the passes shown in Table 2.1.

Assumption 2. The various "passes" tend to be roughly equal in terms of expented effort, because of the tendancy of people to set a series of fairly easily reachable short-term goals for their work. From this it follows that each pass would require roughly the same amount of elapsed time. Time is a more appropriate metric than effort (e.g., man-days), as the requirement assessment tasks tend to be

PASS	ACTIVITY
1	Initial problem discussion.
2	Formal expresison of initial requirements.
3	Assessment of initial requirements, and generation of revised requirements.
4	Initial interdependency analysis - identification of additional errors and inconsistencies.
5	Discussion and generation of revised requirements.
6	Interdependency analysis, decomposition, and determination of initial architecture - discovery of more errors, missing requirements, etc.
7	Final revision to requirements.

Table 2.1

somewhat insensitive to effort level. The key reason for this is that preliminary design is almost always the product of a small number of individuals, frequently a single designer. To the extent that most design-related decisions must "funnel" through a very small cadre, adding additional help doesn't speed things up very much.

Assumption 3. Conceptually at least, there exists from the outset a "perfect" requirements specification - one that specifies all those requirements desired by the system's eventual users, contains no errors, is completely consistent, etc. At any point in time, this perfect set of requirements may be factored into three subsets: a set of recognised requirements, a set of unrecognised by "knowable" requirements, and a set of unrecognised and "unknowable" requirements. The first set includes those requirements that have been correctly elicited. The second set includes requirements waiting to be elicited (if the designers or users could only think of them, they would recognise them as necessary), as well as requirements corrections to previously elicited but incorrect requirements. The third set includes those requirements and corrections that at this time would not be recognised as such even if they were brought to light.

Assumption 4. During requirements analysis step i (see Table 2.1), some proportion p_{e_i} of the remaining knowable requirements "errors" is detected. Here, "errors" should be interpreted broadly, to include incorrect statements,

inconsistencies, missing requirements, etc.

The argument for proportional (as opposed to, say, linear) error discovery rate follows from empirical observation: Bell and Thayer's experiments indicate that, essentially no matter how long and hard requirements are contemplated, there will always be some remaining errors. Error decline seems to be inherently an asymptotically decreasing function of time (i.e., of number of "passes"). Also, the "satisficing" phenomenon first explored by Cyert and March (Cyert and March 64) supports this argument. They pointed out and supported the fact that people generally do not strive for the "very best" in what they do, but rather generally work hard enough at a given task to obtain "satisfactory" results, then rest awhile. In the present context, for instance, a system designer would probably not (during a given pass) seek to determine every last specification error, but rather, having unearthed a certain quantity of such errors, would relax the intensity of his analysis. A "satisfactory" result during each succeeding pass would include a smaller number of specification errors detected than during the preceding pass, which is consistent with the assumption of some proportion of errors detected each pass.

It will be assumed for simplicity that the proportion p_{e_i} for pass i is more or less constant from one pass to another. We will then take p_e as the common proportion of remaining errors turned up during each pass. The nature of

this parameter could be studied empirically in particular design situations.

We may now construct a model of requirements error reduction based on the foregoing assumptions. We let

E_{k0} = the initial level of knowable specification errors, and

p_e = the proportion of errors detected each pass,

then at the end of pass 1 an additional $p_e E_{k0}$ errors will have been determined. Remaining errors are now $E_{k0}(1-p_e)$. Similarly, at the end of pass 2, remaining errors will be $E_{k0}(1-p_e)^2$. In general, at then end of pass i, there will be $E_{k0}(1-p_e)^i$ knowable errors remaining.

Now if we take the unit of time to be days, and assume each pass takes about d_r days, then we may write

$$E_k(t) = E_{k0}(1-p_e)^{d_r t} ,$$

where $E_k(t)$ = the number of knowable errors in the requirements specification at time t.

To complete the error reduction model, a few additional

definitions are required. Let

E_u = the (constant) number of unknowable requirements errors;

s_r = the average staffing level (both users and system analysts) incurred during the requirements analysis and preliminary design phase;

c_r = the average daily cost per staff member employed in this activity (in dollars per man-day);

c_e = the average cost per error still remaining in the requirements set.

The final item, c_e, would probably be difficult to determine directly in practice, although reasonable surrogates are available. Boehm, for instance, has indicated that the cost of fixing errors after a system has been built to run as high as $4000 per line of code. Rough estimates of c_e could be obtained empirically by asking users what they would pay for various functions determined to be necessary after the fact.

Using the foregoing terminology, three equations may be written.

(1) Cost of remaining errors at time t:

$$C_E(t) = c_e[E_u + E_{k0}(1-p_e)^{d_r t}] \quad ...(2.1)$$

(2) Accumulated cost of staff devoted to SDM analysis at time t:

$$C_S(t) = c_r s_r t \quad ...(2.2)$$

(3) Total cost (cost of remaining specification errors plus cost of staff time):

$$C_T(t) = C_E(t) + C_S(t) \quad ...(2.3)$$

A sketch of these curves is given in Figure 2.1.

From this it is possible to define the optimal amount of time, t*, that ought to be devoted to requirements analysis:

Total cost $C_T(t) = C_E(t) + C_S(t)$

$$= c_e[E_u + E_{k0}(1-p_e)^{d_r t}] + s_r c_r t$$

Applying calculus,

$$\frac{dC_r(t)}{dt} = c_e E_{k0}(1-p_e)^{d_r t} \, d_r \ln(1-p_e) + s_r c_r = 0$$

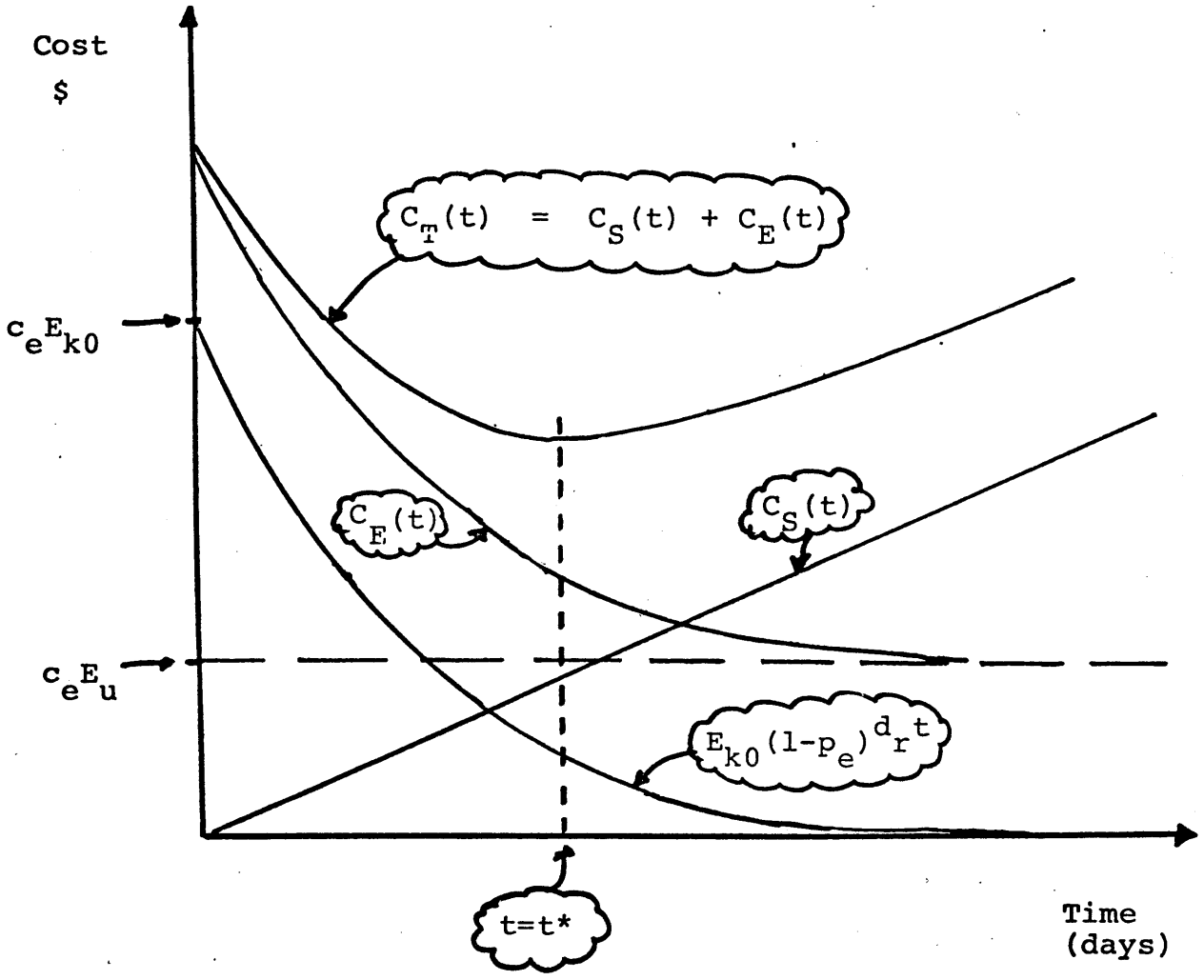

Figure 2.1

Solve for t :

$$t^* = \frac{\ln \dfrac{-s_r c_r}{d_r c_e E_{k0} \ln(1-p_e)}}{d_r \ln(1 - p_e)}$$

The location of t^* is also shown in Figure 2.1.

A reasonable question at this point would be, why we choose to include only the foregoing specification error reduction cost factor in the optimization calculation. The answer is that it is the only payoff factor of the three sub-models being considered here that is significantly time-dependent. Put differently, assuming SDM is used at all, the payoffs to be discussed in Sections 3 and 4 will, in theory, occur. Only the payoff due to specification quality improvement depends, to an important extent, on how much time is spent on the requirements analysis activity.

The error reduction sub-model may be used to show how SDM would improve the requirements analysis phase. In general, the problem with requirements analysis has been that, due to its unstructured nature and due to a lack of user participation, far too little time is usually spent on it in the course of a typical systems development project. In such cases, the model's operating point would lie well to the left of the optimal point t^* for $C_T(t)$. As shown in Figure 2.1 SDM, by structuring the requirements analysis activity and necessitating additional user input, would tend to move the operating point rightward. The value $C_T(t^*)$

specifies a lower bound on the associated cost reduction forthcoming through better preliminary system specification.

The effective dollar benefit from the requirements analysis model may be expressed as

$$B_T(t) = c_e E_{k0} (1 - (1-p_e)^{d_r t}) \qquad ...(2.4)$$

This is just the reflection and shift of the cost-of-knowable-errors curve from Figure 2.1. The earlier analysis may be confirmed by noting that the optimal operating point t^* must occur at the point where marginal cost equals marginal benefit. From before, the marginal cost is just $s_r c_r$. Also, the marginal benefit, from (2.4), is

$$\frac{dB_r(t)}{dt} = -c_e E_{k0} (1-p_e)^{d_r t} \; d_r \ln(1-p_e).$$

Equating these two functions leads to the same result for t^* obtained earlier.

The value of net benefit provided by SDM depends on the actual pre- and post-SDM operating points as well as values of the other parameters. In general, at operating point t, the net benefit will be

$$NB(t) = B_T(t) - C_S(t)$$

$$= c_e E_{k0} (1 - (1-p_e)^{d_r t}) - c_r s_r t$$

Finally, if we assume that SDM application moves the operating point from t' to t, the net improvement in cost will be:

$$I(t',t) = NB(t) - NB(t')$$

$$= c_e E_{k0}[(1-p_e)^{d_r t'} - (1-p_e)^{d_r t}] + c_r s_r (t' - t) .$$

These relationships are sketched in Figure 2.2a,b.

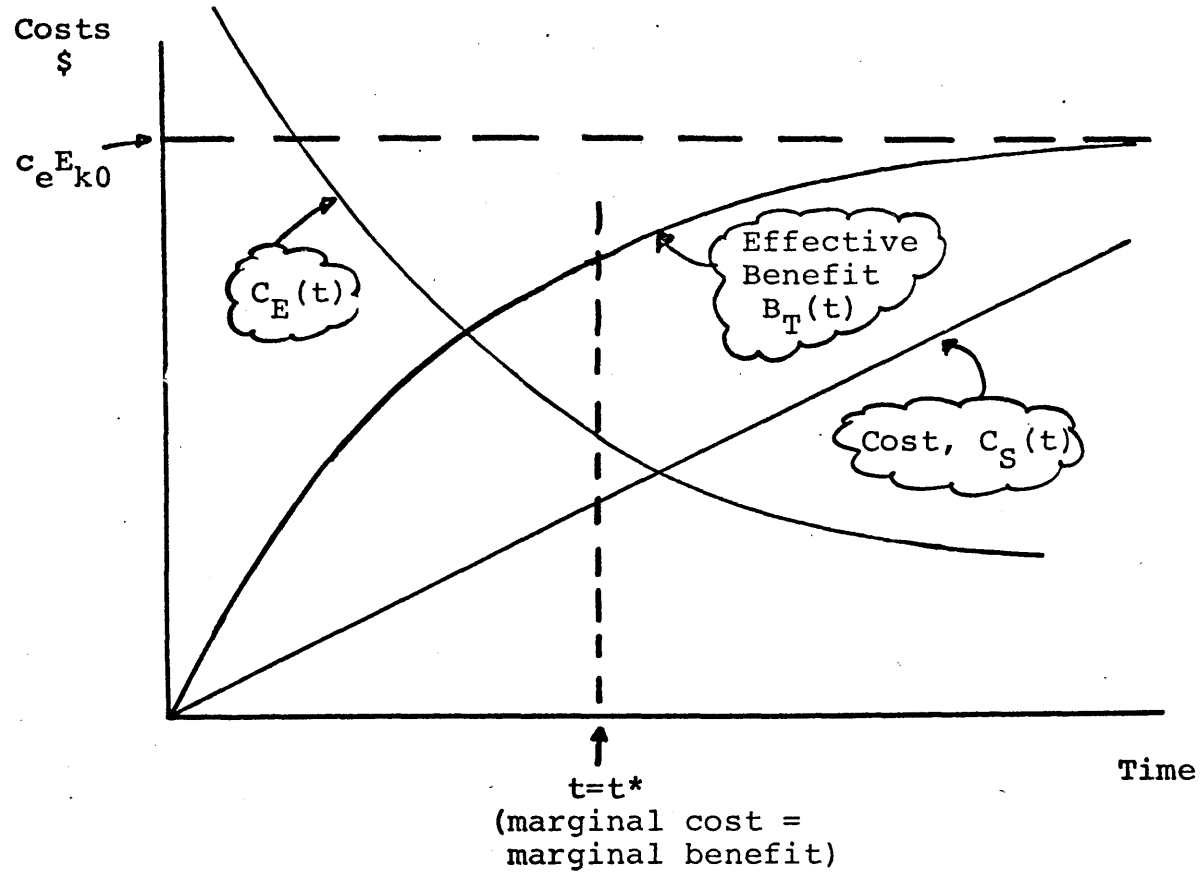

Figure 2.2(a)

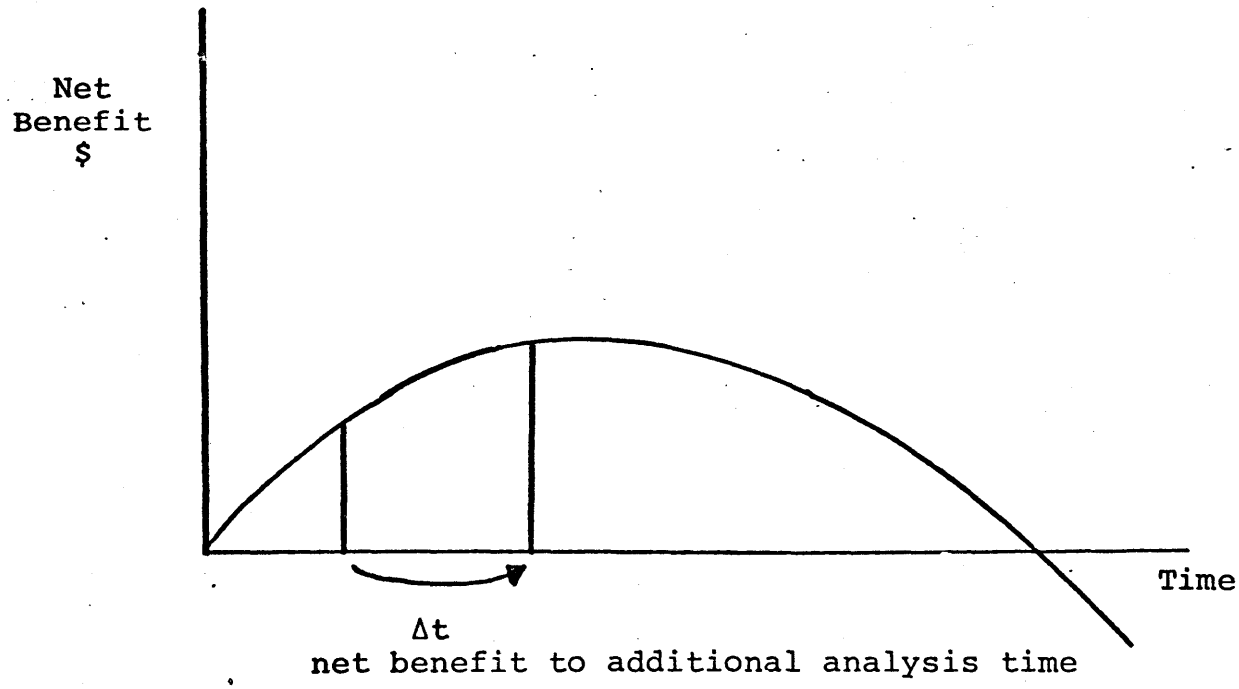

Figure 2.2(b)

3. A System Procedural Development Impact Model.

The second major area of SDM impact concerns the time required to carry out the procedural development (detailed design, programming, and testing) phase of the system life cycle. As a number of authors (e.g., Brooks 75, Scott and Simmons 75) have pointed out, a major impediment to these activities is the need for substantial coordination and communication among the various individuals and groups of people involved. Indeed, Brooks has cited this need as the reason why "men and months are not interchangable" in system procedural development activities.

There are two primary costs that arise as the need for coordination and communication grows:

a) the cost of additional time that the development staff must spend in these activities, which detracts from productive development;

b) the cost of additional layer(s) of project management overhead needed to organize and manage the inter-group coordination and flows of communication.

An example of (a) would be the time spent in intra-group meetings devoted to ironing out issues and misunderstandings that arise from interdependencies between the various tasks. An example of (b) would be the resources consumed in that portion of the formal project management process devoted to managing intra-group issues (e.g., designing a properly

sequenced module testing plan).

In order to see how the Systematic Design Methodology affects the communication and coordination costs of system procedural development, we briefly review some central operational aspects of SDM.

3.1 SDM Operational Aspects.

In its graph partitioning procedure, SDM groups together system functional requirements in such a way as to produce weakly connected system modules. The partitioning objective function, M, consists of a difference between two terms:

$$M = S - C,$$

where S is the sum of the partitioned subgraphs' internal strengths, and C is the sum of the coupling between subgraphs. More specifically,

$$S = \sum_{i=1}^{n} S_i$$

$$C = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} C_{ij}$$

S_i is the strength of module i, and C_{ij} is the coupling between modules i and j. These terms are, in turn, defined out of the requirements graph, and basically involve the normalized richness and strength of the requirement links within a module, and between modules, respectively (see (Huff 79) for more details). While seeking to maximize M is

not exactly the same as minimizing inter-module coupling, the effects are closely related. Identifying modules of high strength tends to arrange requirements that are richly and strongly interconnected together in the same module. In a sense, this is the "dual" objective to minimizing coupling, i.e., to arrange modules so as to minimize interconnectedness between them. Maximizing S and minimizing C thus both drive toward reducing inter-module complexity, in one case by "lumping" complexity inside module boundaries, in the other case by drawing module boundaries so as to keep at a minimum the extent of complexity that is "generated" as a result of the module creation process itself.

3.2 Model Formulation.

Now, we can generalize the notion of a good requirements partition in order to develop a model of the communication/coordination cost impact. We define a variable D as the density of module interconnectedness. A practical measure of D could be the value of system coupling, C, as discussed above. Clearly, the impact of D upon the two factors identified is direct: that is, as D increases, each of these factors will increase also. The graph in Figure 3.1 shows a hypothetical set of curves that depict these relationships.

The first function, relating interconnection density to additional "non-productive" staff time, is illustrated as a

nonlinear function that "blows up" at a certain high level of density. As interconnection density D increases, a larger and larger proportion of staff time must be devoted to the coordination and communication functions, thereby shrinking the time available for productive development (given some initial timetable) toward zero. Conceptually at least, there exists some (high) level of problem complexity, embodied as a large D value, such that the coordination and communication needs among system development staff use up essentially all available time.

Evidence for this phenomenon is cited by Haney (Haney 76) in the context of development and maintenance of a Honeywell operating system, and by Belady and Lehman, in the context of the IBM OS/360 operating system (Belady and Lehman 76).

There is an interesting parallel between the asymptotically growing need for coordination and communication discussed above, and the phenomenon of "thrashing" in a demand-paged virtual memory system. In the latter case, a "high D value" would correspond to a lower level of reference locality for a given set of active processes. A process with a low level of reference locality is essentially one in which the various sub-parts are "tightly interconnected" (in terms of execution references over time), much like a tightly interconnected set of modules in a system architecture that exhibits high D. As locality decreases (D increases), the system overhead costs

rise in a distinctly nonlinear, accelerating fashion, similar to that suggested by Figure 3.1.

In contrast to the first function, the relationship between extra development management overhead and D is hypothesized to be one of linear growth. The evidence for this comes indirectly, from researchers such as Wolverton (Wolverton 78), who suggest that such overhead grows linearly with project size. While D does not necessarily purport to measure size directly, it is reasonable to argue at a first approximation that a doubling of interconnectedness complexity should demand more or less proportionally equivalent management response to that demanded from the size doubling. For example, it requires some, but not a disproportionate amount, of additional project management time to convene a meeting for four team heads as between two, or to send design change documentation to four teams as compared to two.

A simple functional form that may be used to describe the first relationship is

$$S_p(D) = K_p/(D^* - D)$$

where D^* = the density value at which useful staff work becomes impossible,

K_p = a situationally dependent parameter,

$S_p(D)$ = total staff time required for nonproductive work resulting from a level D of logical dependence among design sub-problems (measured in man-days).

Similarly, the functional form for additional project management overhead may be expressed as

$$M_p(D) = L_p D$$

where L_p = a situationally dependent parameter,

and $M_p(D)$ = extra project management time in procedural development necessitated by a level D of logical dependence among design sub-problems (man-days).

It is not suggested that these functions $S_p(D)$ and $M_p(D)$ are fully representative for all values of D. In particular, D values outside the range [0,D] are clearly inapplicable in this model. But even within this range, there exists a smaller "relevant range" within which the functions - especially the first - are really hypothesized to hold. This restricted relevant range is typified in Figure 3.1 with dashed lines.

The first-order impact of SDM on system procedural development time may then be expressed in terms of these

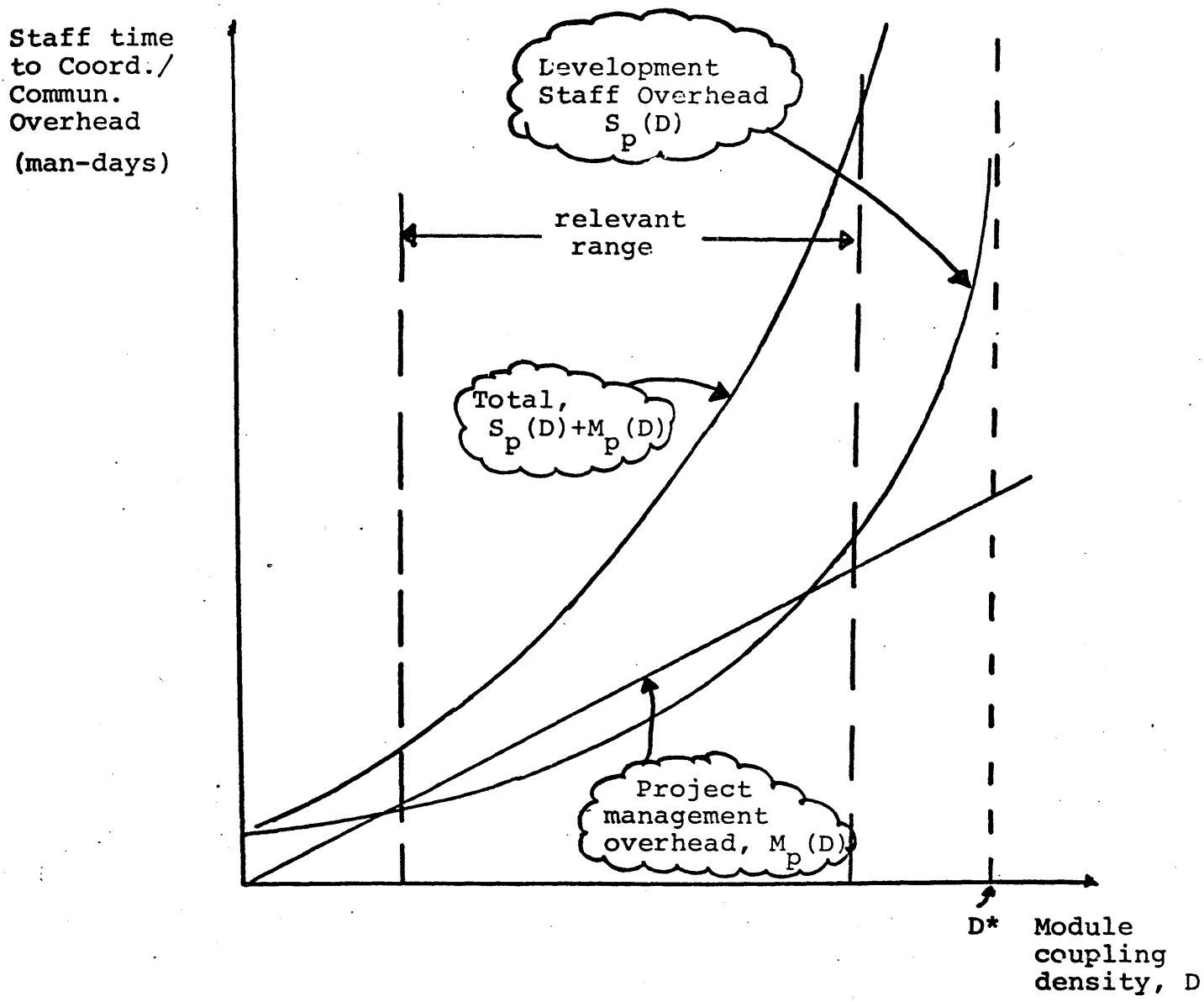
Staff time to Coord./ Commun. Overhead (man-days)
Development Staff Overhead $S_p(D)$
relevant range
Total, $S_p(D)+M_p(D)$
Project management overhead, $M_p(D)$
D*
Module coupling density, D

Figure 3.1

functions. Basically, the effect of SDM is to reduce D for a given project, relative to what it would be if SDM were not used. Suppose the "without-SDM" density value if D_0, and the "with-SDM" value is $D_{SDM} < D_0$. Then the reduction in additional staff time would be

$$\Delta S = K_p \left(\frac{1}{D^* - D_0} - \frac{1}{D^* - D_{SDM}} \right)$$

$$= \frac{K_p (D_0 - D_{SDM})}{(D^* - D_0)(D^* - D_{SDM})}$$

and the reduction in additional project management time would be

$$\Delta M_p = K_p (D_0 - D_{SDM})$$

Then, if C_{Sp} is the average staff cost per unit of time, and C_{Mp} is average project management cost per unit of time, the dollar impact of SDM on development may be expressed as

$$I_p (D_0, D_{SDM}) = C_{Sp} \Delta S_p + C_{Mp} \Delta M_p$$

A slightly different approach to measuring I_p would be to estimate the percentage reduction in D that might occur as a result of using SDM. If experiments were to indicate,

say, $100k_p$ percent impact on D, then the impact on costs of using SDM could be written as

$$I_p(D_0, D_{SDM}) = \frac{C_{Sp}K_pD_0(1-k_p)}{(D^*-D_0)(D^*-D_{SDM})} + C_{Mp}L_MD_0(1-k_p) .$$

Finally, it may be appropriate to rewrite this expression based on D_{SDM} rather than D_0. If we let $100k_p'$ be the percentage amount by which D would increase were SDM not used (assuming, in fact, that it was used), then

$$D_0 = (1 + k_p')D_{SDM} ,$$

and the cost impact expression becomes

$$I_D = \frac{C_{Sp}K_pk_p'D_{SDM}}{(D^*-(1+k_p')D_{SDM})(D^*-D_{SDM})} + C_{Mp}L_p(1+k_p')D_{SDM} .$$

An exercise applying the procedural development impact model is given in Section 5.

4. A System Maintenance/Modification Impact Model.

The third major potential area of impact for the Systematic Design Methodology concerns the cost of maintaining - and, especially, modifying - large software systems. The rapidly inflating costs of system maintenance/modification have been commented upon by numerous authors (e.g., (Boehm 75), (Yau 78)). The need to devise ways of designing and constructing systems so as to reduce the maintenance load and ease the costs of later modification has been widely recognised. One study, for example, has estimated the production cost of a software product to be about $75 per line of code (LOC), while the maintenance cost per LOC of the same system was estimated at $4000 (Boehm 75). A variety of studies have indicated that anywhere from 40 to 80 percent of the original system development cost are eventually spent on "simple" maintenance for the system; when all post-implementation work (including non-trivial modifications) is taken into account the figure rises to 200 to 400 percent (Thayer 77; Goetz 78). At any rate, while often exhibiting rather wide variances, these studies unambiguously indicate that software maintenance and modification functions are assuming increasingly high profiles, and that good system design practice must take this fact fully into account.

It is common practice to differentiate between

"maintenance" and "modification" of software systems. The former term refers to the fairly large number of relatively minor changes, bug fixes and improvements that may be made to a software product following its initial release. Examples include patches to fix minor errors, or the addition of an extra report to a batch-oriented DP system.

In contrast, software modification generally refers to more significant changes made to the system - changes that usually entail some amount of redesign, and that impact most or all of the system's users. Major changes are usually undertaken to add significant new functions to the system, or to improve the operation of a major portion of the system. A prime example is a new release of a vendor's operating system (e.g., IBM's OS/360).

For the purpose of assessing SDM's impact, we will focus primarily on the modification function. More precisely, we will be concerned with those changes to the user-visible functions provided by a particular system - changes which are of significant enough scope that impacts on multiple system modules, or major components, are likely. While arguments could be made that SDM would impact both maintenance and modification costs, the latter impact is almost certainly the more significant.

The primary cause of system modification is the recognition on the part of the user clientele for changes to the functions provided by the system, including such things as addition of completely new functions, major enhancements

to present functions, important improvements in efficiency (response time, turnaround time, etc.), availability, reliability, etc., and changes to fix major system problems. One primary mechanism through which system modifications exert their apparently disproportionately high economic impact is the rippling effect that such changes have on the system. Changes to one system component very often result in the need to make subsidiary changes to other modules. The propagation of these indirect changes results in telescoping of the effect of the original change: a single change can cause other changes, which can in turn cause still other changes, etc. When viewed in this way, the possibility of an instability phenomenon - a single change generating a never-ending sequence of subsidiary changes - presents itself. In fact, evidence exists that such an unstabe situation could occur, and may have been closely approached in certain real-world systems (Haney 76; Belady and Lehman 76).

Since the changes to be implemented are not known a priori, it is not possible to model the change propagation effect deterministically. However, some simple probabilistic arguments provide considerable insight. Consider two system modules, M_i and M_j. Using the SDM framework, these modules are "linked" to the extent that the requirements represented within one module are interdependent with the requirements represented within the other. The greater the number of such interdependency links

relative to the size of the module, the greater the liklihood that a change to one or more of the requirements in the first module will impact (cause a change in) the other.

We define:

r_{ij} = the number of requirements represented in M_i that link to requirements in M_j;

w_{ij} = the average weight on the requirement links between modules M_i and M_j;

α_{ij} = a scale parameter, discussed further below;

n_i = the number of requirements in M_i;

p_{ij} = the probability that a change to M_i necessitates a change in M_j.

Then we propose that, to a first approximation,

$$p_{ij} = \alpha_{ij} r_{ij} w_{ij} / n_i \ . \qquad \ldots(4.1)$$

This relationship presumes that most changes to a module are directed toward a single functional requirement within that module, rarely toward multiple requirements. It also assumes that changes impact each contained requirement in an

equally likely manner. Of course, neither assumption can be strictly true. For instance, certain requirements may be more "central" than others, and thereby stand a relatively greater chance of being impacted. Nonetheless, the representation above is a reasonable first approximation to the system maintenance error propagation phenomenon.

The expression for p_{ij} includes a scaling factor α_{ij}. This factor mainly serves to quantify the effect that not all changes to a function within a given module carry over to other modules, even though its associated requirements may have interdependencies to the other module. While estimating each α_{ij} individually would be difficult, it is quite feasible to estimate an average value, say α, based on the maintenance history for a given design group. Such an approach is followed in the example of Section 5.

According to the above formulation, $p_{ij} \neq p_{ji}$ in general. This is intuitively correct also: for example, if $n_i > n_j$, we would expect there to be a smaller liklihood of a change in M_i impacting M_j than vice versa.

From the above arguments, we can derive the "change propagation liklihood matrix", P:

$$P = [p_{ij}] \quad .$$

The (i,j)th entry in P is the probability that a change to module i necessitates a further (new) change to module j. The change propagation liklihood matrix can now be used to

study the cost impact of system modifications. Suppose at some point in time a set of changes to the system is being considered. Let

a_i = the number of changes to module i being considered.

The term a_i may be thought of as the number of known "bugs" in module i at a point in time (where "bug" is to be broadly interpreted - e.g., an incompletely implemented function demanded by system users would be an example of a "bug"). Also let

$$A = [a_1, a_2, \ldots, a_n] .$$

A is the vector of planned module changes; A will be termed a "revision" to the system.

Now, the originally planned changes $a_1, a_2, \ldots, a_n$ give rise to additional changes. If, for instance, there are a_j planned modifications to module j, this will generate $a_j p_{j1}$ expected number of changes to module 1, $a_j p_{j2}$ changes to module 2, and so forth. The overall expected impact of the planned changes - the average number of "second-level" changes - is simply the matrix product AP. The first element of the row vector AP is the expected number of second-level changes to module 1, etc.

The second-level changes themselves give rise to yet

additional changes, in a corresponding manner. The expected number of third-level changes to each module can be calculated as

$$(AP)P = AP^2.$$

Conceptually, this telescoping of module changes continues ad infinitum. Mathematically, the total expected number of changes to each module may be calculated by summing the resulting infinite series,

$$\begin{aligned} T &= A + AP + AP^2 + AP^3 + \ldots \\ &= A(I + P + P^2 + \ldots) \end{aligned} \qquad \ldots(4.2)$$

If all the eigenvalues of the matrix P are real and lie in the range (-1,1), then a basic result in matrix algebra (Strang 77) says that this matrix series converges, to the value

$$T = A(I - P)^{-1} . \qquad \ldots(4.3)$$

Essentially, expression (4.3) summarizes the ripple effects that occur as a result of modifications to modules of a large system.

This basic model of module connectivity can be now used to model the impact of SDM on system maintenance. First of all, the essential effect of SDM decomposition analysis is

to determine an optimal, or nearly optimal, decomposition of the system requirements with respect to the decomposition objective function, M. As discussed in Section 3, this leads to requirements partitionings with minimal inter-partition coupling.

We will assume that, as a first-order result, the coupling between requirements represented within modules i and j is reduced by a percentage $100\delta_{ij}$% as a result of using SDM. Alternatively, the inter-module liklihood of change propagation, p_{ij}, is changed to a new value $p'_{ij} = \delta_{ij} p_{ij}$. Thus the new change propagation liklihood matrix is then

$$P_{SDM} = \begin{bmatrix} \delta_{11}p_{11} & \cdots\cdots & \delta_{1n}p_{1n} \\ & & \\ \delta_{n1}p_{n1} & \cdots\cdots & \delta_{nn}p_{nn} \end{bmatrix}$$

If we further assume for a moment the special case wherein all δ_{ij} are equal to a common value δ, then we have

$$P_{SDM} = \delta P .$$

Now, if the series

$$I + P + P^2 + \ldots \qquad \ldots(4.4)$$

converges, then so does the series

$$I + P_{SDM} + P_{SDM}^2 + \ldots \qquad \ldots(4.5)$$

The proof of this follows.

Proof. Since we assume that the first series converges, the eigenvalues of P must all be real and lie in the range (-1,1). Now, the eigenvalues of P are the values of λ that solve the equation

$$\left|P - \lambda I\right| = 0 . \qquad \ldots(4.6)$$

On the other hand, the eigenvalues of P_{SDM} are solutions to the equation

$$\left|P_{SDM} - \lambda_{SDM} I\right| = 0 , \text{ or}$$

$$\left|\delta P - \lambda_{SDM} I\right| = 0 .$$

This is equivalent to

$$\left|\delta \left(P - \frac{\lambda_{SDM}}{\delta} I\right)\right| = 0 , \text{ or}$$

$$\left|P - \mu I\right| = 0 , \qquad \ldots(4.7)$$

where $\mu = \lambda_{SDM} / \delta$. But equation (4.7) is identical to (4.6) above. Consequently, $\lambda = \lambda_{SDM}/\delta$, or $\lambda_{SDM} = \lambda\delta$.

Since $|\delta| < 1$ and $|\lambda| < 1$, it follows that $|\lambda_{SDM}| < 1$. But this is just the necessary and sufficient condition that series (4.4) above converges.

- - - - -

Therefore it is clear that the effect of reducing all the module change propagation liklihood values can only serve to reduce the ripple effect and make the modification process converge faster.

Now we define the average cost of making a change to module i as c_i . The vector of such costs is

$$C = (c_1 , c_2 , c_3 , \ldots , c_n) .$$

The total cost of all first-order changes may then be expressed as

$$C_1 = AC_1 = a_1c_1 + a_2c_2 + \ldots + a_nc_n .$$

The value C_1 may be viewed as the "optimistic" cost - i.e., the cost of carrying out a revision (set of changes) to a software system under the (naive) assumption that $P = 0$.

Alternately, if the ripple effect is taken into account, the total cost C_T of the revision is expressed as

$$
\begin{aligned}
C_T &= C_1 + C_2 + C_3 + \ldots \\
&= AC + APC + AP^2C + \ldots \\
&= A(I + P + P^2 + \ldots)C \\
&= A(I - P)^{-1}C \quad . \qquad \ldots(4.8)
\end{aligned}
$$

This view of the cost of the revision A might be termed the "realistic" cost, as opposed to the foregoing "optimistic" cost, in that it takes into account the ripple effect.

Finally, the "realistic" cost of the revision A under the assumption that SDM was employed would be

$$C_T(SDM) = A(I - P_{SDM})^{-1}C \ ,$$

where $P_{SDM} = (p'_{ij}) = (\delta_{ij} p_{ij})$ as defined earlier.

Hence the cost reduction for the revision A that could be attributed to SDM would be:

$$
\begin{aligned}
C_T &- C_T(SDM) \\
&= A(I - P)^{-1}C - A(I - P_{SDM})^{-1}C \\
&= A\left[(I - P)^{-1} - (I - P_{SDM})^{-1}\right] C \\
&= ABC \qquad \ldots(4.9)
\end{aligned}
$$

where we define $B = (I - P)^{-1} - (I - P_{SDM})^{-1}$.

Some insight can be gained into the nature of this function by considering some special cases. Suppose the change propagation liklihood value p_{ij} was

$$p_{ij} = \begin{cases} p_i & \text{if } i = j, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Then a single modification made to any one module results in a total number of expected changes to that module of

$$1 + p_i + p_i^2 + \ldots = 1/(1 - p_i)$$

For instance, if there is a 10% chance of a change in module i resulting in another change in the same module, then $p_i = 0.10$, and the expected number of changes resulting from a single change to module i would be 1.1111... .

Now assume that there are n modules, and $p_{ij} = p$ for all i,j . That is, the liklihood of change propagation between any pair of modules if 100p%. Then a single original change to one module generates np expected second-order changes, each of which generates another np expected third-order changes, etc. The total number of changes is then

$$1 + np + (np)^2 + (np)^3 + \ldots = 1/(1 - np) .$$

This expression reflects the impact of both change propagation liklihood p and system size, in terms of number of modules, n.

There is an asymptotic limit at the point np = 1, or

$p = 1/n$, as shown in Figure 4.1. This limit represents the point at which the system is large enough and interconnected enough that a single change will ripple forever throughout the system - the number of generated changes "blows up." This blow-up phenomenon is closely related to the point $D = D^*$ discussed in the Procedural Development Model of Section 3.

The foregoing analysis indicates the large payoff to reducing the degree of interconnection of a system's design, even by a small amount. Haney (Haney 75) points out that informal experiments with the Xerox Universal Timesharing System showed each change to be causing approximately 10 additional changes. This system had about 22 major modules, so that

$$1/(1 - np) = 1/(1 - 22p) = 10 + 1 = 11 .$$

Hence $p = 0.04$. If the interconnection propagation liklihood could have been reduced by 25%, to $p = 0.03$, the number of additional changes following a single change would have dropped from 11 to 3! This would obviously lead to substantial savings in maintenance costs.

Additional illustration and siscussion of this and the earlier models is given in the following section.

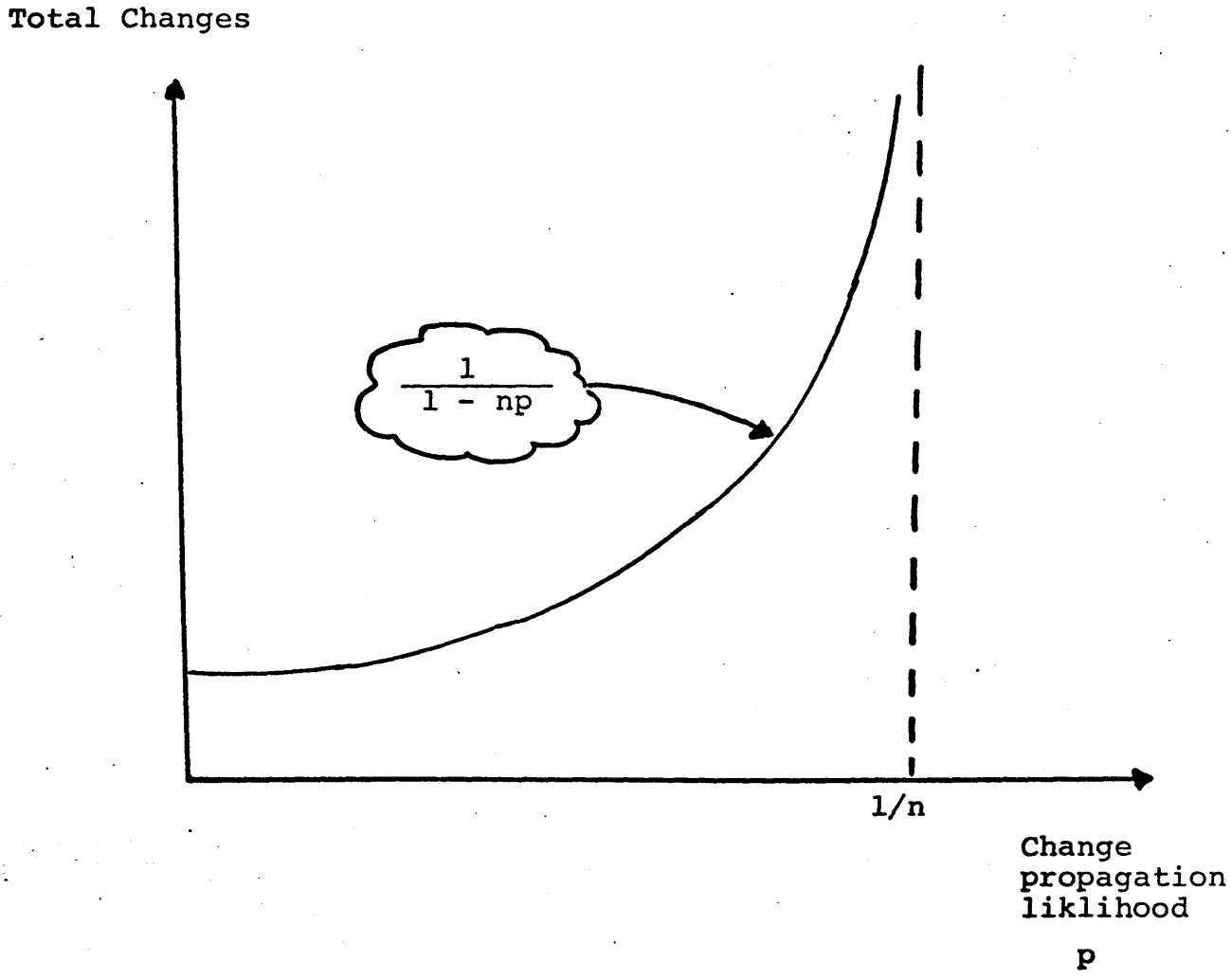

Figure 4.1

5. An Illustration.

5.1 Recapitulation.

In the previous three sections, three different models, one for each of the major impact areas of the Systematic Design Methodology, were described. To recap briefly, the three areas are:

a) system specification - correctly identifying and stating as many of the "knowable" functional requirements as possible, within the constraints of time and manpower available;

b) system procedural development - carrying out the detailed design, coding, and testing of the system, together with the accompanying load of coordination and communication overhead among members of the development team(s);

c) system maintenance/modification - making necessary or requested alterations to the functions provided by the system - correcting errors, adding new functions, etc. - while simultaneously making sure that all secondary, tertiary, etc. changes to other system modules are also identified and carried out.

The three models developed in the previous sections are different from each other in certain ways. For one thing, the first two are deterministic, while the third contains probabilistic elements. Also, in the first model, time is the key independent variable, whereas in the second the focus is on the density of module interconnectedness, and in the third, the number of changes to be made to different system modules together with the probabilities of change propagation. All three models attempt to capture the effects of the modelled independent variables and parameters

on cost - i.e., various components of cost are the dependent variables in the different models.

5.2 Example Calculation.

In order to further illustrate the functioning of the SDM impact models, a sample calculation for a hypothetical system is presented in this section.

Consider a system design problem with the following characteristics:

a) There are 200 functional requirements making up the system's user-level functional specifications;

b) The preliminary design has factored the requirements into 10 partitions, each containing an average of 20 requirements.

c) In the base case (pre-SDM) partitioning, on the average, any given functional requirement is interdependent with 10 other requirements. The average interdependency strength factor is 0.60.

We will consider each of the three impact models in turn.

5.2.1 Specification Impact (Model 1).

To carry out computations using this model, we need numerical values for the terms E_{k0}, E_u, p_e, d_r, s_r, c_r, and c_e. The following values will be taken as reasonable estimates for present-day technology and salary scales:

PARAMETER	VALUE	UNITS
E_{k0}	50	knowable errors (initial level)
E_u	20	unknowable errors
p_e	0.2	(unitless - proportion of errors detected per pass)
d_r	3	days/pass
s_r	20	man-days/day (average staff level)
c_r	100	dollars/man-day
c_e	1000	dollars/error

With these values, the equations for model 1 are

$$C_S(t) = s_r c_r t = 2000t \qquad ...(5.1)$$

$$C_E(t) = c_e[E_u + E_{k0}(1 - p_e)^{d_r t}]$$
$$= 1000[20 + 50(1 - 0.2)^{3t}] \qquad ...(5.2)$$

$$C_T(t) = C_S(t) + C_E(t) \qquad ...(5.3)$$

These three equations are sketched in Figure 5.1

Further calculations show the optimal time spent in SDM analysis is

$$t^* = \frac{\ln\left[\frac{-s_r c_r}{d_r c_e E_{k0} \ln(1-p_e)}\right]}{d_r \ln[1-p_e]}$$

$$= 5.2 \text{ days} . \qquad ...(5.4)$$

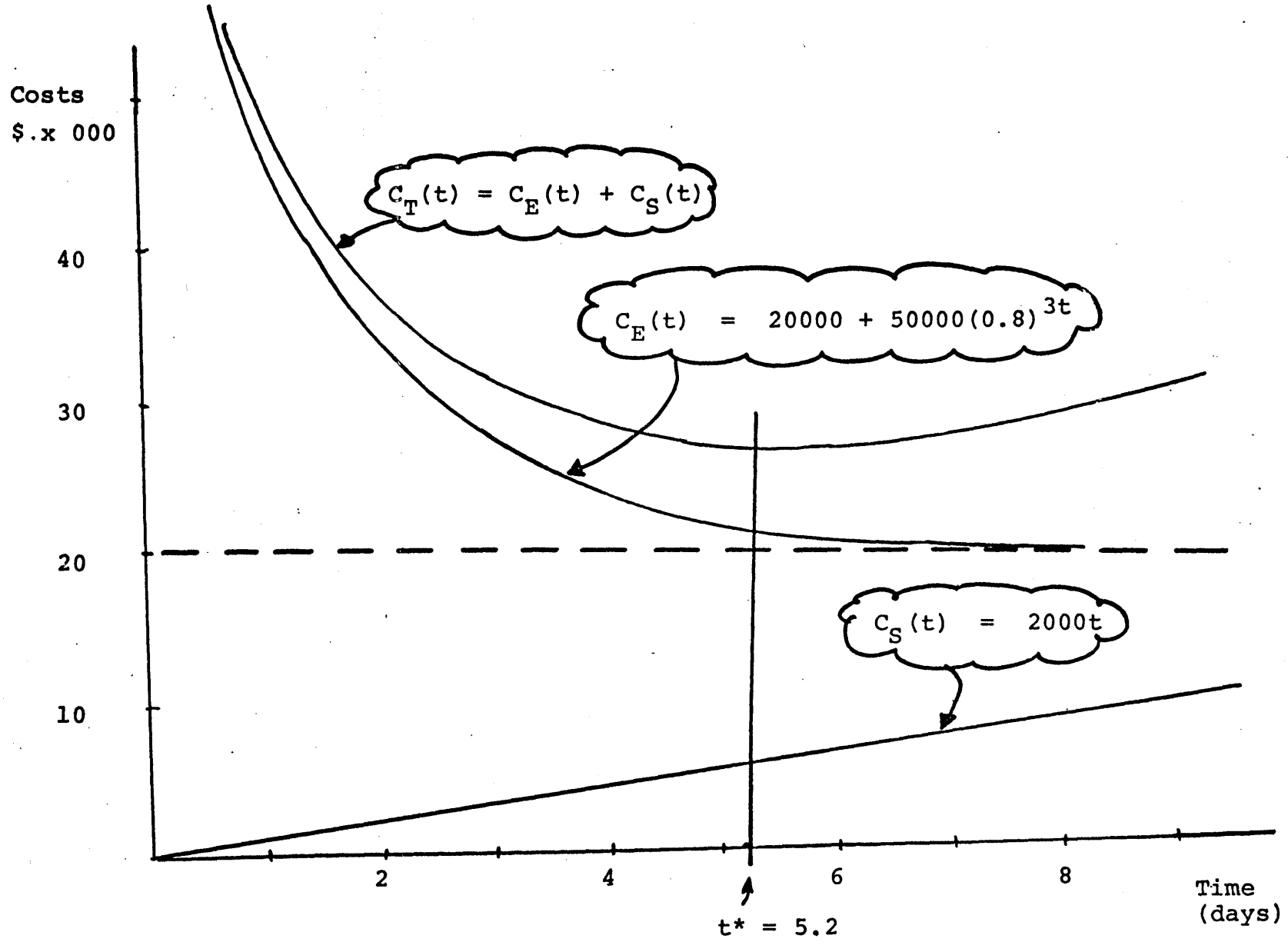
Costs
$.x 000
40
30
20
10
$C_T(t) = C_E(t) + C_S(t)$
$C_E(t) = 20000 + 50000(0.8)^{3t}$
$C_S(t) = 2000t$
2
4
6
8
Time
(days)
t* = 5.2

Figure 5.1

Thus, approximately two SDM passes are suggested as best for minimizing total cost (requirements errors plus staff time). The optimal t value is also indicated on the sketch in Figure 5.1.

The resulting cost of analysis plus remaining requirements errors is calculated as approximately $26,700 at $t = t^*$. The net improvement at $t = t^*$ as compared with $t = 0$ (no SDM analysis done) is

$$I(0,t^*) = c_e E_{k0} [1 - (1-p_e)^{d_r t}] + c_r s_r t$$

$$= 50000(.969) + 10400 = 58860$$

5.2.2 Procedural Development Impact (Model 2).

If we now turn to examine the second model for this problem, we see that we need estimates for the following terms: D^*, K_p, L_p, D_{SDM}, D_0, C_{Sp}, and C_{Mp}. Consider first the D terms. In Section 3 we did not constrain the units of D specifically, but merely defined D to be a measure of module interconnectedness. For the present purposes we will take D to be the per-module sum of the interdependency weights on all the external (inter-module) requirement interdependencies. This is an intuitively easy value to work with, and demands no further assumptions regarding the structure of the requirements graph (e.g., module coupling estimates).

Now, using the data given earlier, each of the ten modules contains (an average of) 20 requirements. Each requirement links to ten other extra-module requirements, with an average link weight of 0.60. Therefore,

$$D_0 = (20 \text{ requirements/module}) \times (10 \text{ links/requirement}) \times (0.60 \text{ strength units/link}).$$
$$= 120 \text{ (strength units/module)}.$$

For illustrative purposes, we will now assume that SDM analysis produces a superior partitioning of the requirements, and results in each requirement linking to six other extra-module requirements, with an average link weight of 0.40. Therefore

$$D_{SDM} = (20 \text{ req./module}) \times (6 \text{ links/req.}) \times (0.4 \text{ strength units/link})$$
$$= 48 \text{ strength units/module.}$$

Next, we need an estimate of D^*. A possible approach to estimating the complexity limit D^* would be to consider the worst possible case - i.e., the case in which every requirement linked to every other extra-module requirement with a maximum weight. This would give rise to a D^* value of $(20) \times (9 \times 20) \times (1) = 3600$. However, such an extreme case is clearly not representative of realistic settings. At issue regarding D^* is the question of how complex, in terms of

module interrelationships, a real design scenario needs to be before essentially all useful effort ceases - i.e., before "thrashing" fully sets in. To draw on the virtual memory analogy again, thrashing can completely bottle up a virtual storage computer system long before the theoretical worst case (a page fault, or multiple page faults, on every new instruction - see (Madnick and Donovan 76, page 498)) occurs. Therefore, for present purposes let us assume that the "thrashing" level of D is $D^* = 200$. In other words, we will assume here that, if inter-module complexity were roughly twice as high as in the base design case, almost all designer time would be spent in coordination and communication, with little or no project headway possible.

To calculate the parameter K_p, we would need to estimate the per-man project overhead under D = 0. This corresponds to each module design being independent of all others. Under such conditions, there would still be some basic level of general coordination and planning overhead to be incurred. For argument's sake we take this to be 50 days per man over the duration of the project.

Under this assumption, at D = 0, we have

$$S_p(0) = K_p/D^* = 50 \text{ days/man, so}$$
$$K_p = 50(200) = 10{,}000.$$

The units of K_p are (staff days/man)x(weight units/module).

Finally we will estimate L_p by assuming that under the

base case design about 100 project management man-days would be required. Therefore,

$$L_p = 100/120 = 0.83 .$$

The units of L_p are (management days/man)x(modules/weight unit).

We take the per-diem staff and project management costs as before to be

$$C_{Sp} = 100 \text{ dollars/man-day, and}$$
$$C_{Mp} = 150 \text{ dollars/man-day} .$$

With the foregoing data, the model's equations are:

$$S_p(D) = K_p/(D^* - D) = 10000/(200 - D) \qquad ...(5.4)$$

$$M_p(D) = L_p D = 0.83D \qquad ...(5.5)$$

The cost impact curves for Model 2 are given in Figure 5.2. The points D_0 and D_{SDM} are indicated, as are the associated cost impacts.

The cost savings under the foregoing assumptions and

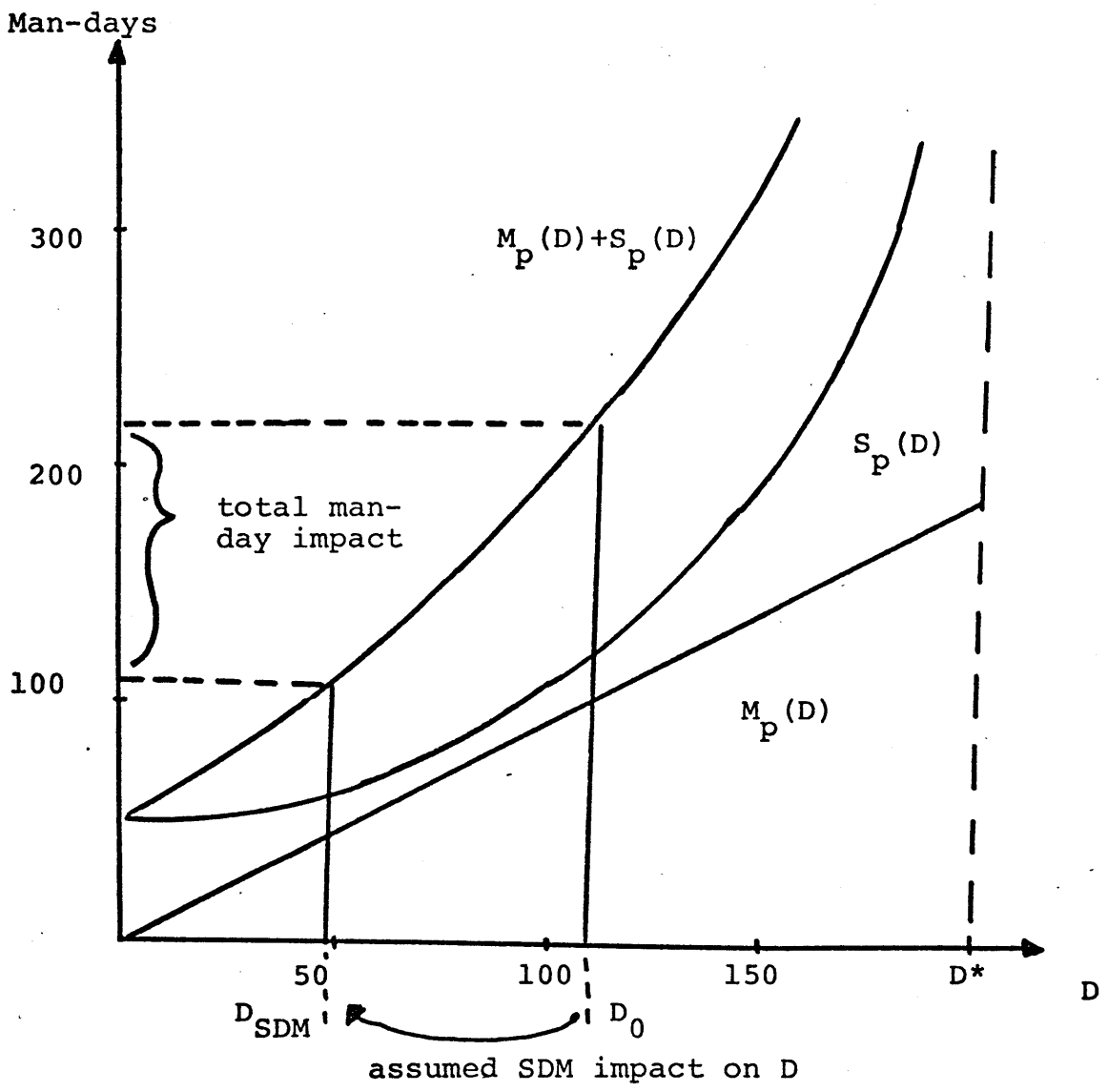
Man-days
300
200
100
$M_p(D)+S_p(D)$
$S_p(D)$
$M_p(D)$
total man-day impact
50
100
150
D*
D
D_{SDM}
D_0
assumed SDM impact on D

Figure 5.2

estimates turn out to be:

$$I_p = C_{Sp} S_p + C_{Mp} M_p = \frac{100(10000)(120-48)}{(200-120)(200-48)} + 150(.83)(120-48)$$

$$= 5921 + 8964 = \$14,885$$

Therefore, use of the design methodology reduces impact of development overhead under the given assumptions by about 15,000 dollars.

5.2.3 Modification/maintenance Impact (Model 3).

Now we turn to the third model, impact on modification/maintenance. This model, being probabilistic and matrix-oriented, is computationally somewhat more complex.

In the present case, if we consider a typical module pair, using the information provided earlier, we find that equation (4.1) becomes

$$p_{ij} = \alpha_{ij} 10(.6)/20 = (0.3)\alpha_{ij} \quad , i \neq j \ .$$

Now we will estimate the α_{ij} scale factor at 0.25, for all i,j. Consequently we take each

$$p_{ij} = (0.3)(0.25)\alpha = 0.075 \quad (i \neq j).$$

Also, we will estimate the on-diagonal probability to

be twice this value, 0.150. Thus the matrix P has the form:

$$P = \begin{bmatrix} .15 & .075 & .075 & \cdots & .075 \\ .075 & .15 & .075 & \cdots & .075 \\ \vdots & & & & \\ .075 & .075 & .075 & \cdots & .15 \end{bmatrix}$$

Of course, in a more realistic situation the P matrix would be sparser; there would be a number of module pairs with no interdependencies. The above calculation is quite reasonable for illustrative purposes.

To continue the example, we will also estimate the δ-factors as

$$\delta = 0.67 .$$

That is, we assume that partitioning the requirements via SDM reduces the inter-module change propagation probabilities by 1/3. Thus the new P matrix becomes:

$$P_{SDM} = 0.67(P)$$

$$P_{SDM} = \begin{bmatrix} .10 & .05 & .05 & \cdots & .05 \\ .05 & .10 & .05 & \cdots & .05 \\ \vdots & & & & \\ .05 & .05 & .05 & \cdots & .10 \end{bmatrix} .$$

A check shows that both P and P_{SDM} do have all their eigenvalues in the (-1,1) range, so their series do converge. The values of these matrices are given in Table 5.1.

To complete the example using Model 3, let us assume that each module will require two significant modifications over the lifetime of the system. Therefore we let

$$A = [a_1, a_2, \ldots a_n] = [2, 2, 2, \ldots, 2] .$$

Also, we will assume that the average cost for modifying a module is 1000 dollars, i.e.,

$$C = [c_1, c_2, \ldots, c_n] = [1000, 1000, \ldots, 1000] .$$

Under these assumptions, the "optimistic" modification cost is

$$C_1 = AC = \$40,000$$

The "realistic" cost for the given revision is, however,

$$C_T = A(I - P)^{-1} C$$
$$= \$109,841 .$$

Hence the SDM modification cost reduction for this revision

$$I + P_0 + P_0^2 + P_0^3 + \ldots = (I - P_0)^{-1}$$

$$= \begin{bmatrix} 1.544 & .463 & .463 & \ldots\ldots & .463 \\ .463 & 1.544 & .463 & \ldots\ldots & .463 \\ \vdots & & & & \\ .463 & .463 & .463 & \ldots\ldots & 1.544 \end{bmatrix}$$

(a)

$$I + P_{SDM} + P_{SDM}^2 + P_{SDM}^3 + \ldots = (I - P_{SDM})^{-1}$$

$$= \begin{bmatrix} 1.170 & .117 & .117 & \ldots\ldots & .117 \\ .117 & 1.170 & .117 & \ldots\ldots & .117 \\ \vdots & & & & \\ .117 & .117 & .117 & & 1.170 \end{bmatrix}$$

Table 5.1

would be

$$ABC = 2(1000)\underline{1}\ B\ \underline{1}^t$$

where $B = (I - P_0)^{-1} - (I - P_{SDM})^{-1}$

and $\underline{1} = [1\ \ 1\ \ 1\ \ \ldots\ 1]$ (10 elements).

Calculations show that

$$ABC = 2(1000)\ \underline{1}\ B\ \underline{1}^t$$

$$= 2000\ [1\ 1\ \ldots\ 1] \begin{bmatrix} .375 & .346 & .346 & \ldots & .346 \\ .346 & .375 & .346 & \ldots & .346 \\ & & & & \\ .346 & .346 & .346 & \ldots & .375 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \\ 1 \end{bmatrix}$$

$$= 69841\ .$$

By this estimate, SDM partitioning would lead to an estimated cost savings over the modification life of the system of nearly $70,000.

5.3 Summary of Illustration.

The results of the calculations in the foregoing example are summarized in the table below.

ITEM	RESULT
Optimal SDM Analysis Time	5.2 days
Cost of Analysis (at t = t*)	$10,400
Specification Error Reduction Benefit (at t = t*, as compared to t = 0)	$58,860
Base Case Module Interconnectedness (D_0)	120
SDM Module Interconnectedness (D_{SDM})	48
Development Benefit	$14,885
Modification "Ripple Effect" Benefit	$69,841
Total SDM Benefit	$143,586
Total SDM Cost	$10,400
Net SDM Benefit (this example)	$133,186

Table 5.2

6. Conclusions.

The models presented and discussed here are in many ways crude and of limited scope. But, as we pointed out earlier, there is no other published work on modelling the impacts of design methodologies at all. This preliminary effort at least illustrates the feasibility for improving understanding of the economic potential of improved functional design partitionings and other related aspects of systems design.

A number of issues were raised but not completely answered in this report. Some of the most important are:

- how to accurately determine the various parameters of such models;
- how to better verify the correctness of the models' functional forms;
- how to make effective use of such models - e.g., what can be learned from sensitivity analysis or parametric variation?
- how to improve the models - e.g., perhaps we ought to be looking at different aspects; perhaps some of the "second-order" effects are really more important than they were assumed to be here.

These and other questions will be studied in future investigations.

REFERENCES

Andreu, R.C., and S. E. Madnick: "Completing the Requirements Set as a Means Towards Better Design Frameworks: A Follow-up Exercise in Architectural Design", Internal Report P010-01-06, MIT Sloan School of Management, December 1977.

Andreu, R. C.: A Systematic Approach to the Design and Structuring of Complex Software Systems, PhD. Thesis, Sloan School of Management, M.I.T., Cambridge, Mass. 1978.

Aron, J.: "Estimating Resources for Large Programming Systems", in Buxton, J., and J. Randell (eds.): Software Engineering Techniques, NATO Science Committee, Brussels, 1970.

Aron, J.: The Program Development Process - Part II: The Development Team, preliminary draft version, Addison-Wesley Publishers, June 1977.

Belady, L., and M. Lehman: "A Model of Large Program Development," IBM Systems Journal, vol. 15, no. 3, 1976.

Bate, R., and G. Ligler: "An Approach to Software Testing: Methodology and Tools," Proc. Third Conference on Software Engineering, I.E.E.E. Publications, 1978.

Bell, T., and T. Thayer: "Software Requirements: Are They Really a Problem?", Proc. 2nd Int. Conf. on Soft. Eng., 1976.

Boehm, B.: "Software and Its Impact: A Quantitative Assessment", Datamation, vol. 19, no. 5, May 1973.

Boehm, B.: "Some Steps Towards Formal and Automated Aids to Software Analysis and Design", Information Processing 74, North-Holland, 1974.

Brooks, F.: The Mythical Man-Month, Addison-Wesley, 1975.

Cave, W., and A. Salisbury: "Controlling the Software Development Cycle - The Project Management Task," IEEE Trans. on Soft. Eng., vol. 4, no. 4, July 1978.

Champine, G. A.: A Total Life Cycle Cost Model for a Computer System, PhD Thesis, University of Minnesota, 1975.

Cooper, J.: "Corporate Level Software Management," IEEE Trans. on Soft. Eng., vol. 4, no. 4, July 1978.

Cyert, R., and J. March: A Behavioral Theory of the Firm, Prentice-Hall, 1963.

Davis, C., and C. Vick: "The Software Development System", IEEE Trans. Soft. Eng., vol. 3, no. 1, Jan. 1977.

Dolotta, T., et. al.: Data Processing in 1980 - 1985, Wiley-Interscience, 1976.

Ginsberg, M.: "A Detailed Look at Implementation Research," MIT Sloan School Center for Information Systems Research, WP 753-74, 1974.

Goetz, M.: "The Software Products Industry - Its Future and Promise," Computerworld, In-Depth Report, October 17, 1978.

Haney, F.: "The Architecture of Software," Data Base, ACM SIGBDP Newsletter, 1976.

Huff, S. L., and S. E. Madnick: "An Approach to the Construction of Functional Requirement Statements for System Architectural Design", Internal Report No. P010-7806-06, June 1978(a).

Huff, S. L., and S. E. Madnick: "An Extended Model for a Systematic Approach to the Design of Complex Systems", Internal Report No. P010-7806-07, July 1978(b).

Huff, S.: "Decomposing Weighted Graphs Using the Interchange Partitioning ATechnique," Technical Report 8, MIT Sloan School of Management, January 1979(a).

Huff, S.: "Analysis Techniques for Use With the Extended SDM Model," Technical Report 9, MIT Sloan School of Management, February 1979(b).

Kustanowitz, A.: "System Life Cycle Estimation," Proc. Third Int. Conf. on Software Engineering, I.E.E.E. Publications, 1978.

Leintz, B. et. al.: "Characteristics of Application Software Maintenance," Comm. of the ACM, vol. 21, no. 6, June 1978.

Madnick, S. E., and J. Donovan: Operating Systems, McGraw-Hill 1975.

Pietrasanta, T.: "Resource Analysis of Computer Programming System Development," in On The Management of Computer Programming, G. Weinwurm (ed.), Auerbach Publishers, 1970.

Putnam, L.: "A General Empirical Solution to the Macro Software Estimating and Sizing Problem," IEEE Trans. on Soft. Eng., vol. 4, no. 4, July 1978.

Ross, D., and K. Schomann: "Structured Analysis for Requirements Definition", IEEE Trans. Soft. Eng., vol. 3, no. 1, Jan. 1977.

Scott, R., and D. Simmons: "Predicting Program Development Productivity - A Communications Model," Proc. First Nat. Conf. on Soft. Eng., IEEE Publications, Sept 1975.

Stay, J. F.: "HIPO and Integrated Program Design", IBM System Journal, vol. 5, no. 2, June 1976.

Stevens, J., et. al.: "Structured Design", IBM Systems Journal, vol. 13, no. 2, April 1974.

Strang, G.: Linear Algebra and Its Applications, Academic Press, 1977.

Teichroew, D., and M. Bastarache: "PSL Users' Manual", ISDOS TR No. 98, March 1975.

Thayer, T.: "Understanding Software Through Empirical Reliability Analysis," Proc. of the NCC, 1975.

Watson, C., and C. Felix: "Method of Program Measurement and Estimation," IBM Systems Journal, vol. 16, no. 1, 1977.

Wasserman, T., et. al.: "Software Engineering: The Turning Point", Computer, September 1978.

Wolverton, R.: "The Cost of Developing Large-Scale Software," IEEE Trans. on Computers, vol. 23, no. 6, June 1974.

Yau, S., et. al.: "Ripple Effect Analysis of Software Maintenance," Trans. Third Int. Conf. on Soft. Eng., IEEE Publications, 1978.