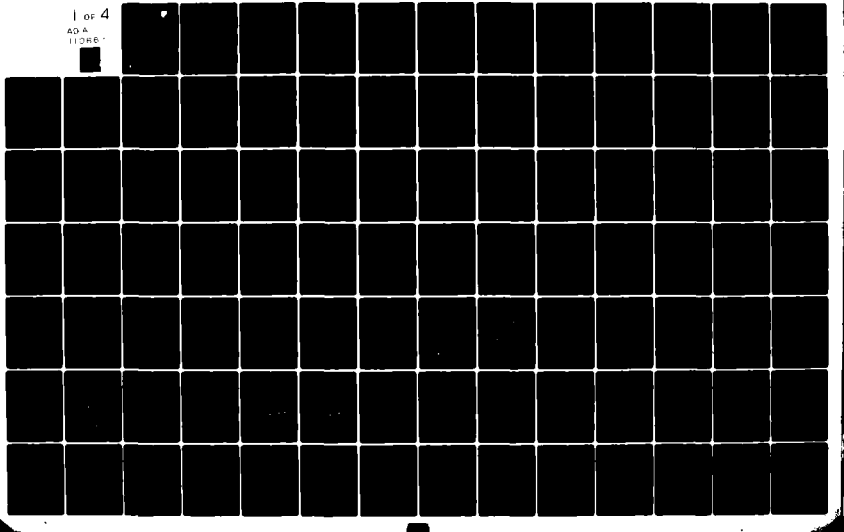
AD-A110 867 INCO INC MCLEAN VA F/G 9/2
SMALL SCALE SYSTEMS.(U)
SEP 81 M KERCHNER, P GRIMES F30602-80-C-0219
UNCLASSIFIED INCO/1155-681-TR-46-D(F) RADC-TR-81-251 NL

1 OF 4
AD A
110867



(12) LEVEL II

RADC-TR-81-251
Final Technical Report
September 1981

SMALL SCALE SYSTEMS

INCO, Inc.

Dr. Marcia Kerchner
Peter Grimes

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-81-251 has been reviewed and is approved for publication.

APPROVED: John J. Maier

JOHN J. MAIER
Project Engineer

APPROVED: John N. Entzminger Jr.

JOHN N. ENTZMINGER, JR.
Technical Director
Intelligence & Reconnaissance Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRDT) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-81-251	2. GOVT ACCESSION NO. AD-A110 867	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SMALL SCALE SYSTEMS		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report 15 Jun 80 - 15 Jun 81
		6. PERFORMING ORG. REPORT NUMBER INCO/1155-681-TR-46-D(F)
7. AUTHOR(s) Dr. Marcia Kerchner Mr. Peter Grimes		8. CONTRACT OR GRANT NUMBER(s) F30602-80-C-0219
9. PERFORMING ORGANIZATION NAME AND ADDRESS INCO, Inc. 8260 Greensboro Drive McLean VA 22102		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 45941663
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (IRDT) Griffiss AFB NY 13441		12. REPORT DATE September 1981 13. NUMBER OF PAGES 304
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

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)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: John J. Maier (IRDT)

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

Scale System	Scaling Parameters
Performance Indices	Scaling Metrics
Simulator Variable	Individual Walston and Felix Group
Life-Cycle Estimation Models	Component Rates
Scaling Freedom	

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

This report defines the applications of scaled systems as design instruments for designing, developing, and evaluating intelligence systems, in order to provide a concrete means of investigating and ascertaining the various factors pertinent to the application of scaled systems.

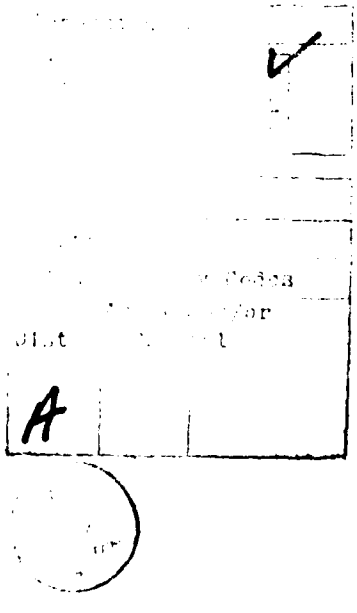
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)



LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (CONTINUED)

SECTION 1. INTRODUCTION

The cost of developing large scale computer systems has increased dramatically in the last few years. In spite of more sophisticated design techniques, many systems fail to meet cost, schedule, cost-benefit, and performance objectives; many systems, once completed, do not perform as well as they are expected to; many systems never even get completed!

Current software system development methodologies emphasize a process in which development is conceived as proceeding through a series of phases. Each phase is organized to complete a specific planned process and produces output in terms of information or design documents that are input to the next phase. Most attempts to improve the effeciency of the development cycle have concentrated on improving the processes which comprise some single phase. Structured programming focuses on the programming stage of the development phase while composite design applies to the design stage of the development phase.

There is a need, however, for design validation at less than full-system cost, and for prototyping design alternatives. The use of integrated scaled systems presents such a technique.

Scaled systems are operational systems implementing subsets of capabilities and/or performance characteristics of the ultimate full-scale system. The scaled system approach is intended to bridge the gap between the definition and design stages of the development phase.

Using scaled system concepts for the design, development, and evaluation of intelligence data handling computer systems is expected to improve the way these tasks are performed. By implementing a subset of

the capabilities of a full-scale system, a "scaled system", it is anticipated that the initial expenditure on the scaled system, a fraction of the cost of the full-scale one, will decrease the overall full-scale system cost, schedule, and risk. Because scaled systems are operational systems, users can immediately obtain the benefits available from partial automation of their requirements.

The use of scaled systems within a development effort can have several benefits. However, only some of the benefits may be applicable to any specific development effort. Which of the benefits are desirable will determine the objectives for using scaled systems within the development effort. Once these objectives are established, the precise manner in which the scaled system should be defined from the full-scale one can be determined. Knowledge of benefits realizable from the application of scaled systems is therefore vital to understanding the scaled system technique, so potential benefits are listed below.

a. Users can begin using a scaled system as soon as it is implemented, since scaled systems are operational systems. Feedback from users can guide final design decisions for the full-scale system. This benefit is particularly important in instances where users are unable to clearly specify their requirements for automated support due to their lack of experience with computers or to the unique nature of the tasks they desire to automate. The scaled system can be used to demonstrate exactly what capabilities are available to the user as well as give the user an idea of how he will interface with the system and what procedures must be developed. Based on his experience with a scaled system, the user will then be able to clearly specify his requirements for the

full-scale system, thereby greatly increasing the probability of success for the overall development effort.

b. Different techniques for performing unique or state-of-the-art operations can be tried with scaled systems, in order to establish feasibility of complex designs or to determine the optimal way to provide certain capabilities within different environments.

c. The team developing a scaled system obtains valuable experience with the project that increases their productivity when developing the full-scale system. Design lessons learned from the scaled system also decrease the number of false starts and blind alleys encountered during full-scale development.

d. In many instances a scaled system can be incrementally expanded to eventually implement the desired full-scale system. The incremental development approach is usually more cost-effective than is an attempt to implement an entire large-scale system at once in a turnkey fashion.

e. The cost and schedule for scaled system development, once that development is complete, can be used as a predictor for the cost and schedule of full-scale system development. This effort has examined how reliable predictors can be established.

f. The performance of a scaled system can be used as a predictor for the performance of the corresponding full-scale system. Full-scale systems often fail to meet their performance objectives, and the use of a scaled system may indicate that a redesign, increase of system resources, and/or relaxation of performance objectives is required to achieve the desired full-scale system performance. In cases where the scaled system indicates that the desired performance is achievable, the performance

predicted by the scaled system can be used in the evaluation of the full-scale system ultimately implemented, thereby reducing the risk of implementing systems with inadequate performance. This effort has researched the development of reliable performance predictors for scaled systems.

A scaled system is implemented during definition or design stages in the life cycle of full-scale system development. The scaled system may be developed based on the functional description for the full-scale system, or, in certain instances, based on the system specification. It is desirable to implement the scaled system as early in the development cycle as possible, as experience gained with the scaled system can provide valuable insight for later full-scale system design. Thus, the preferred approach is that the scaled system be implemented based on the full-scale functional description, and that the full-scale system specification be developed based on the scaled system. It should be noted that the scaled system has its own development cycle similar to that of the full-scale system, except with much shorter schedules.

The scaled system originally implemented as a design tool can then be used again during the evaluation phase of the full-scale system developent cycle. This research has investigated techniques for predicting full-scale system performance based on scaled system performance. Thus, measurements made on the scaled system can originally be used to predict full-scale system performance, and can later be used to evaluate how well the implemented full-scale system achieved those performance predictions.

Section 2 discusses the research methodology, including the objectives of the effort and how they were achieved, in particular, in terms of the simulator and cost model developed in this effort. Section 3 describes the specific results of the research, including the definitions of scale factor metrics, system parameter interrelationships, guidelines on scaling system scale factors, decision factors and guidelines indicating when to use scaled systems as part of a design effort, and anticipated cost benefits of employing scaling techniques. Section 4 discusses research efforts that will be fruitful areas for further investigation.

SECTION 2. METHODOLOGY

The objective of this effort was to conduct research to define the applications of scaled systems as design instruments for designing, developing, and evaluating intelligence systems, in order to provide a concrete means of investigating and ascertaining the various factors that are pertinent to the application of scaled systems. Various elements of software systems, "System Scale Factors," were evaluated with the specific objective of identifying the elements most suitable to small scaled applications. These items were then quantified to provide a uniform and standardized terminology allowing objective categorization of scaled systems, based on the corresponding full-scale system. In order to determine the interrelationships among these scale factors, so that they may be considered in the overall system methodology for using scaled systems, a concept was evolved that uses a simulation model of a generalized IDHS to predict performance and to predict changes in one scale factor variable from changes in another.

Scaled systems techniques were developed to provide better estimates of total development cost, schedule, and performance, by defining decision factors for using scaled systems, in order to indicate when scaled systems should be used as part of a design effort. The decision factors are to provide justification in terms of ultimate full-scale system cost, schedule, risk, and performance, for using a scaled system. A preliminary integrated cost model, synthesizing the best characteristics of the models studied into a single model suitable for scaled systems research, was implemented and calibrated with data derived from analysis of an actual intelligence system, the Defense Intelligence

Agency (DIA) Integrated Indications System (DIIS), in order to place the decision factor guidelines on a firm quantitative footing.

The objective was then to identify specific benefits realizable from the scaled systems approach by analyzing past systems developed and comparing an actual scaled system to its full-scale counterpart, namely the NMIC system and INCO's scaled version of the NMIC's User Support Subsystem (USS) called the Indications and Warning Training System (IWTS). These two systems (NMIC and IWTS) were compared and contrasted in terms of their relative size, cost, hardware configuration, software implementation, complexity, difficulty, and effort expended to complete them, as far as the data permitted such analysis.

Section 2.1 describes the design of the overall effort. Section 2.2 describes the operating system performance simulator and Section 2.3 the cost models designed for evaluation of actual and proposed scaled systems.

2.1 General

In order to define the scaled system methodology, two similar types of relationships were considered in this effort: (1) how the performance of a scaled system compares to that of a full-scale system and (2) how the use of a scaled system affects the total cost, schedule, and risk of a system development effort. The first type of relationship is required to predict the performance of a full-scale system based on that of a scaled system, while the second type of relationship is required to judge the benefit of using a scaled system as part of a development effort. Both types of relationship, taken together, are also required to determine precisely which system parameters should be scaled,

and by what amount, to take maximal advantage of a scaled system within a given development effort.

The way in which the elements of the technical approach combine to satisfy the total research objectives can be summarized as follows:

- Identify system parameters that are suitable for scaling.
- Define scale factors for each of these parameters.
- Examine the correlations and interrelationships among scale factors.
- Use these correlations for developing guidelines of which parameters to scale and how much, based on system objectives.
- Prepare a list of decision factors that are indicative of whether or not scaled systems should be used as part of a development effort.
- Develop guidelines for whether or not scaled systems should be used based on these decision factors.
- Identify specific benefits realizable from the scaled systems approach for past systems developed and future systems to be developed.
- Quantify benefits for planned systems realizable through the use of scaled systems.

2.2 The INCO System Performance Simulator

The INCO system performance simulator (ISPS) is an event-driven simulator designed to execute on INCO's interactive microprocessor-based computer systems. The simulator models a generalized, variable computer system configuration consisting of a CPU, a disk, a user-specified number of on-line terminals, and the associated system queues necessary to simulate the allocation of these resources. A detailed abstract technical discussion of the simulator can be found in Appendix F and discussions concerning its operation and method of application to this

research can be found in the earlier portions of this section. The objective of this discussion is to highlight the simulator's functional characteristics.

The simulator was designed in a programmer's design language (PDL) and subsequently coded into FORTRAN. Its purpose, as previously stated, was to model a variable computer system environment. This variable environment is specified by the simulator's user by way of a description of the system configuration's component characteristics. These input parameters are specified by the user at run-time through an interactive query. The input parameter set and its format is illustrated in Figure 2-01. This is the same query the user iterates through before simulator execution.

During the simulation, the user may optionally observe the steps the simulator makes through a video display that is updated by the simulator at the occurrence of each new simulator event. The execution speed of the simulator is increased, however, if the user selects the "truncated" terminal display format as opposed to this "extended" format which requires the additional processing overhead of the terminal I/O in order to periodically update the display. The screen display is illustrated in Figure 2-02. A sample simulator performance output is shown in Figure 2-03.

Using this simulator, an analyst can explore the rudimentary performance characteristics of varying computer system hardware configurations as well as the effects of generalized job-type mixes. Job types are classified as either CPU- or disk-bound for purposes of the simulation. For example, the simulator can help the analyst determine

SPSIM INPUT PARAMETERS

(a) Number of terminals?
(b) % Percentage mix between CPU- & Disk- bound jobs?
(c) Mean CPU service time (CPU bound jobs)?
(d) Mean CPU service time (Disk bound jobs)?
(e) Mean Disk service time (CPU bound jobs)?
(f) Mean Disk service time (Disk bound jobs)?
(g) Mean CPU/Disk iteration count (CPU bound jobs)?
(h) CPU/Disk iteration count std. dev. (CPU bound jobs)?
(i) Mean CPU/Disk iteration count (Disk bound jobs)?
(j) CPU/Disk iteration count std. dev. (Disk bound jobs)?
(k) Mean wait time for terminal # <1-100>?
(l) Std. dev. about wait time for terminal # <1-100>?
(m) Extended(F) or Truncated(T) Screen Display?

Figure 2-01 Simulator Input Parameter Set

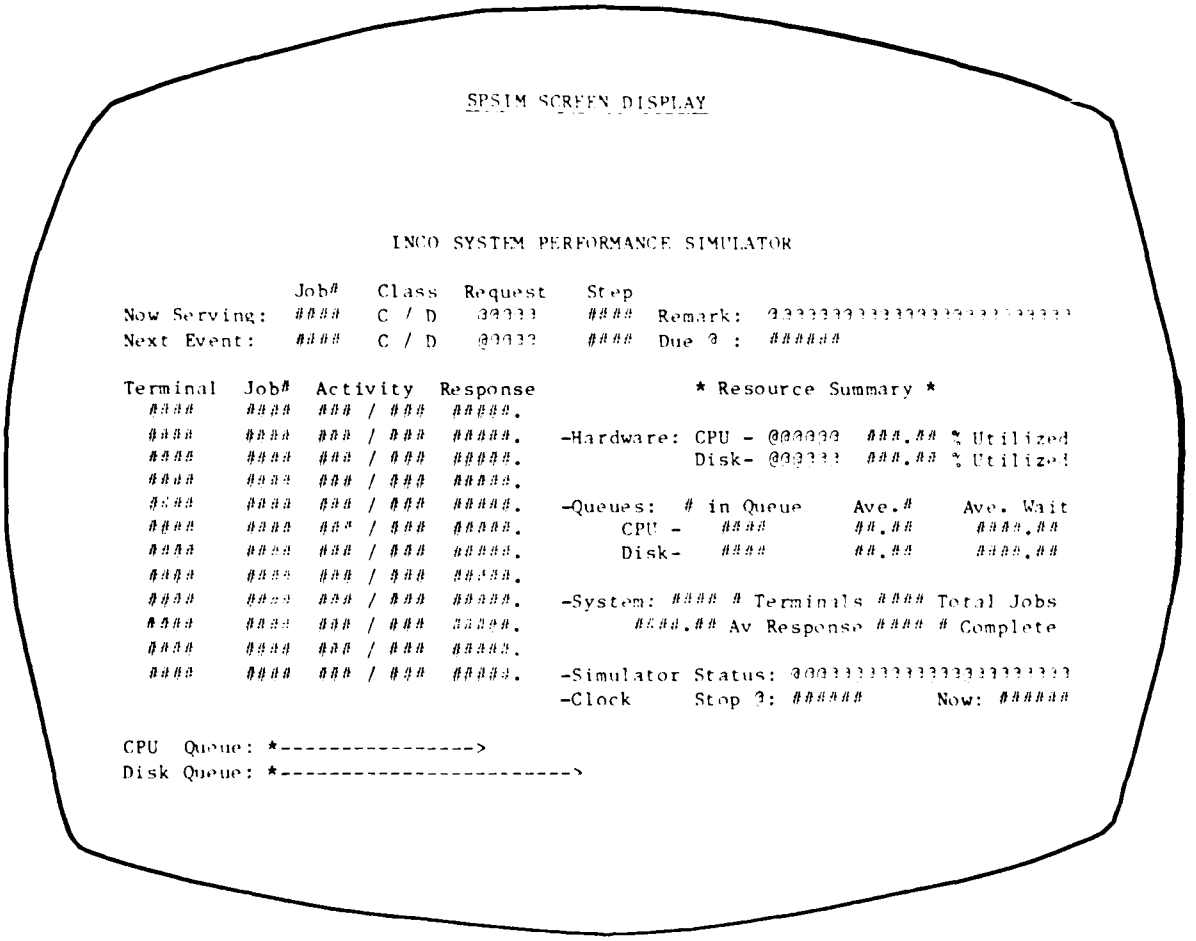


Figure 2-02. Simulator Screen Display

INCO SYSTEM PERFORMANCE SIMULATOR RESULTS

Comment: RUN #69
Time and date of run: 11:36-APRIL 27, 1981

INPUT PARAMETER SUMMARY

	MEAN VALUE	STAND.DEM.
Service times:	1.25	0.00
CPU:		
CPU - Bound jobs	1.25	0.00
Disk - Bound jobs	1.40	0.00
Disk:		
CPU - Bound jobs	35.00	0.00
Disk - Bound jobs	40.00	0.00
Iteration counts:		
CPU - Bound	10.00	1.00
Disk - Bound jobs	30.00	3.00
Terminal Delay Times:		
Terminal # 1	50.00	0.00
Terminal # 2	50.00	0.00
Terminal # 3	50.00	0.00
Terminal # 4	50.00	0.00
Terminal # 5	50.00	0.00
Terminal # 6	50.00	0.00
Terminal # 7	50.00	0.00
Terminal # 8	50.00	0.00
Terminal # 9	50.00	0.00
Terminal #10	50.00	0.00
Number of On-line terminals:	10.	
Job Mix (ratio of CPU/Disk bounds jobs):	50.00	

Figure 2-03. Sample Simulator Performance Output

INCO SYSTEM PERFORMANCE SIMULATOR RESULTS

Comment: RUN #61
Time and date of run: 11:36-APRIL 27, 1981

SIMULATION RESULTS

Terminal Responsiveness:

Terminal #	Jobs Queued/Completed	Average Response
1	6/ 6	3571.81
2	7/ 7	3292.74
3	4/ 4	5699.30
4	5/ 5	4626.85
5	8/ 8	2878.71
6	7/ 7	3086.97
7	6/ 6	3879.37
8	5/ 5	4509.58
9	6/ 6	3505.37
10	7/ 7	2968.22

System Performance Summary:

Number of Terminals	=	10
Number of Jobs Submitted	=	61
Number of Jobs Completed	=	61
Elapsed Time	=	23276.
Average Responsiveness	=	3650.56
Hardware Utilization:		
CPU -	=	3.48%
Disk -	=	99.93%

Queue Summary	Average # in Queue	Average Wait Time
CPU -	.00	1.08
Disk -	8.41	322.90

Figure 2-03. (Continued)

the relative impacts of such system configuration changes as the addition of on-line terminals, faster or slower terminals, faster or slower disks, or a CPU with different speed characteristics. Internally, the simulator considers only a single CPU and a single disk; this does not present a major problem, however, as multiple devices can be accounted for by assumptions concerning their service time efficiencies. For example, adding disk drives and/or controllers can be reflected through a decrease in the disk service time parameter of the input mix, which has the effect of speeding up the simulation of disk I/O. Additionally, many general system performance characteristics can be observed or validated through the use of this simulator. For example, use of the simulator has reflected the hypothesis that the responsiveness of computer configurations is limited by the slowest memory present in the configuration, namely the auxiliary disk storage. Because of this, it can be witnessed that the disk resources are heavily utilized in terms of the usage of their available time. Simulations consistently showed that the disk resources were 90-100% utilized, whereas the CPU was only 3-25% utilized. Through the use of this simulator, the interrelationships of scaled system configuration items could be examined.

2.3 The INCO Cost Estimation Model

The INCO life cycle cost model is the result of extensive research performed in the areas of software engineering, life cycle software cost estimating, and scaled system development by INCO, INC. The model is the reflection of INCO's commitment to develop a low-cost software life cycle cost model for in-house use on low-cost microprocessor hardware.

2.3.1 Genesis

INCO began research and development of its own software life cycle costing model in May of 1979. The first step of this effort included training sessions with the PRICE S and SLIM software cost estimating models and the start of what would become an intensive literature search and study. In this phase, INCO personnel absorbed as much as was possible from available information on the subjects of software cost estimation, commercial software cost models, and software life cycle cost behavior and management. Published research which was found to be of most value is summarized in Figure 2-04. A comparison of open-literature models was performed, and an example is included in Figure 2-05.

Along that point in time, some of INCO's other contracted-for research efforts realized the need for some sort of cost estimation tool, however rudimentary. One such effort was the Scaled Systems Project.

Under the Scaled Systems effort, INCO was providing research support to the Rome Air Development Center (RADC) in the way of exploring cost effective software development methodologies, particularly in the areas of prototype and scaled/prototype developmental systems. As part of this effort, critical cost relationships between scaled systems and their full-scale counterparts were examined. Of specific interest were the potential benefits which could be derived from the experience an organization would gain from the implementation of a scaled operational version of a state-of-the-art system before actual development commenced on the full-scale system. Of additional interest was the sensitivity of the forecasted cost benefits to overall scale factor. This was the first application of INCO's cost model. To explore the productivity and cost

COST MODEL DEVELOPMENT

STEP 1: SURVEY OF PUBLISHED RESEARCH

- DOTY & ASSOCIATES
- IBM'S WALSTON & FELIX
- IEEE'S TUTORIALS ON SOFTWARE COSTING
- MAURICE HALSTEAD'S "SOFTWARE SCIENCE"
- UNIVERSITY OF MARYLAND'S COMP. SCI. DEPT. (VIC BASILI)
- DACS'S SURVEY OF SOFTWARE COST ESTIMATING MODELS
- LAWRENCE PUTNAM (SLIM)
- ISPA'S NEWSLETTER AND PROCEEDINGS
- RCA'S PRICE-S

Figure 2-04. Significant Cost Model Literature

PROGRAM SIZE (1,000's DSLOC)	IBM's Walston & Felix	SDC	Putnam	Doty (Application = All)	Halstead Estimator	Modified Halstead Estimator	μ σ σ/μ % for 1,2,3,&4
1	5.2	5.5	4.9	5.3	1.5	.5	5.2 .22 4%
5	22.5	23.7	23.4	28.8	44.9	7.7	24 2.4 10%
10	42.3	44.3	46.0	60.0	194.3	24.4	48 6.9 14%
50	182.8	189.7	221.2	328.6	5741.3	348.1	230 58 25%
100	343.6	355.0	435.0	683.6	24506.5	1082.4	454 137 30%
250	790.9	812.6	1063.9	1800.7	165,999.0	4818.0	1,117 409 37%
500	1486.2	1520.2	2092.6	3746.6	703,112.0	14,804	2,211 918 42%
750	2149.4	2193.0	3108.3	5751.2	1,633,630	28,500	3,300 1,465 44%
1000	2792.6	2844.2	4115.8	7795.1	2,969,540	45,327	4,386 2,037 46%

Figure 2-05. Comparison of Open Literature Models Effort Results

impacts of such factors as realizing personnel experience and firmness of operational requirements, the Doty model was exercised about varying system sizes in the context of developing full-scale systems from built-to-scale systems. A sample of the model's interactive display used for such analysis is provided in Figure 2-06. This figure reveals the cost factors accounted for by the Doty model. The generalized result from the scale factor sensitivity analysis is portrayed in Figure 2-07.

Such exercise proved invaluable to the development of the cost model. After initial survey and exercise of current cost modeling methodologies, INCO adopted the approach of synthesizing the best characteristics of each model it had scrutinized into the one model. The theoretical basis, however, remained close to the properties outlined by Lawrence Putnam in his many research works. These remaining steps of model development are summarized in Figure 2-08.

2.3.2 Foundation.

The basic Putnam model (Figure 2-09) was attractive for a number of reasons. First, it is the best of the "publicized" models - its internal characteristics are defined, outlined, and validated in print. The internals of a model such as PRICE S, in contrast, are very closely held by its inventors and vendor, RCA. Second, the Putnam model has the best facilities for adaptability and changeability through its technological constant and software equation. Third, the Putnam methodology seems to be the best accepted, on a theoretical basis, and many other researchers are actively exploring its properties, behavior, and possibilities. Fourth, the possibilities the Putnam model holds as a tracking/management tool looked promising. This was especially important for an ancillary

From the Doty & Associates (RADC) Studies:

```
                Please Select an Application Category:

                    1 - Utility (OS)
                    2 - Command & Control (c2)
                    3 - Scientific
                    4 - Business
                    5 - All (Others not listed above)

                Selection (1-5)?
                Estimated Deliverable Source LOC (1,000's)?
                (S)cale, (U)pscale, or (O)ption? O

    Please input a yes/no (Y/N) response to each of these 14 questions:

     Special display?
     Detailed definition of operational req'mts?
     Change to operational req'mts?
     Real time operation?
     CPU memory constraint?
     CPU time constraint?
     First S/W developed on CPU?
     Concurrent development of ADP H/W?
     Time share, vis-a-vis batch processing, in dev'ment?
     Off-site development computer facilities?
     On-site development computer facilities?
     Development computer different than target computer?
     Multi-site development computer facilities?
     Unlimited programmer access to computer facilities?

9999.99 Man Months req'd for analysis, design, code, debug, test and checkout.
            ( Standard error on this approximation = 99.9 % )

                Estimated schedule duration = 999.99 Months

                 Continue (Y or N)?
```

Figure 2-06. Example of INCO Model's Interactive Display

COST / BENEFIT ANALYSIS (TYPICAL)

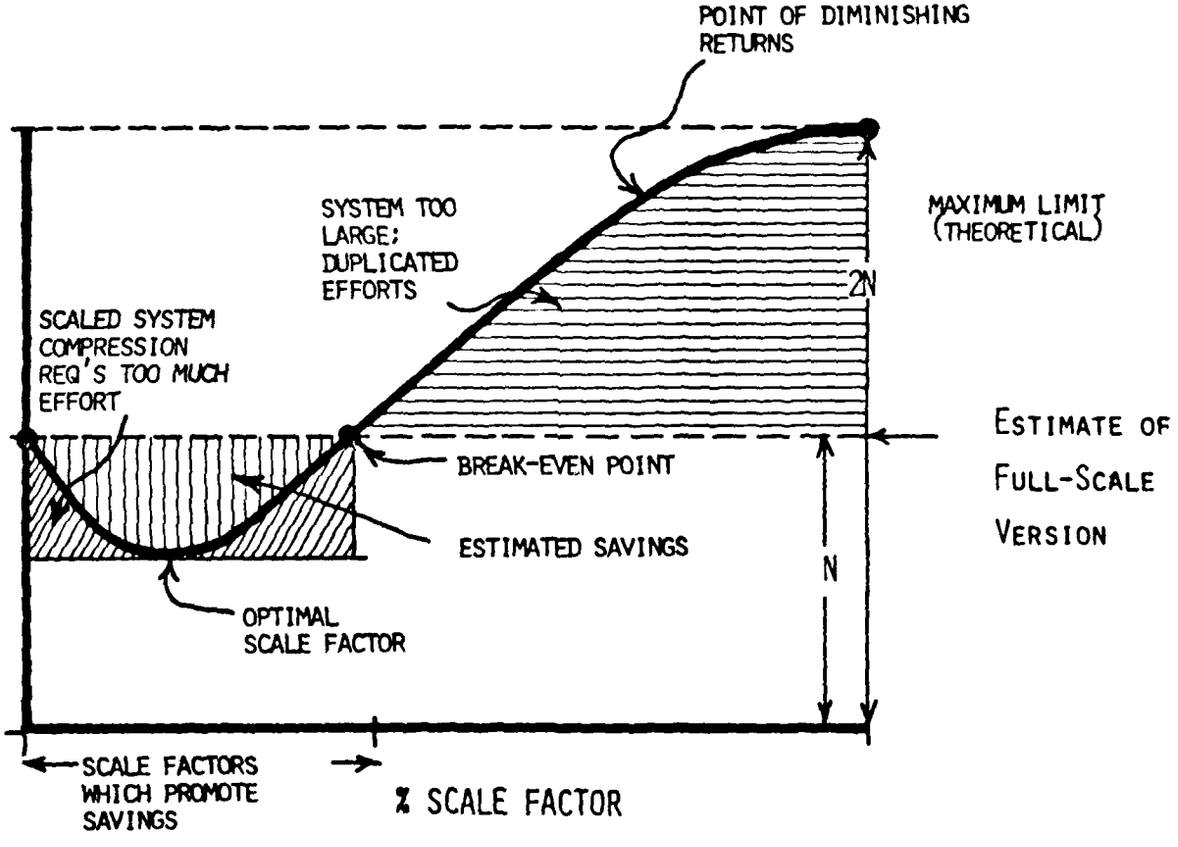


Figure 2-07. Scale Factor Sensitivity Analysis

STEP 1:

- Program Generalized Cost Formulas in BASIC
- Exercise & Compare Results
- Tech. Memo; "Scaled Systems Cost Effectiveness"

STEP 2:

- Putnam Methodology Selected As Most Suitable For Our Purposes
- Began Detailed Implementation & Development

STEP 3:

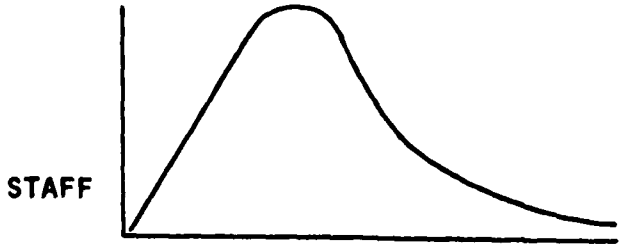
- Began Calibration of Model to Other Models and Past Experience

Figure 2-08. INCO Cost Model Development

PUTNAM MODEL

BASIS:

$$\dot{y} = 2Kate^{-at^2}$$



THE RAYLEIGH/NORDEN CURVE

THE "MANAGEMENT" PARAMETERS:

SCHEDULE	(t_d)
EFFORT	(K)
STAFF	($\dot{Y}$)

THE SOFTWARE EQUATION:

$$S_s = C_k K^{1/3} t_d^{4/3}$$

THE CRITICAL "TECHNOLOGY CONSTANT"
- CHARACTERIZES ENTITIES
- ACCOUNTS FOR DIFFERENT PRACTICES
- SUBJECT OF CALIBRATION

Figure 2-09. Relationships in Basic Putnam Model

effort taking place at INCO that consisted of the design and development of a integrated set of individual models addressing the entire scope of software development. This effort is highlighted by the automated implementation of INCO's tried and proven requirements Structured Organization and Analysis Procedure (SOAP) -- namely, the Requirements Analysis and Tracking System (RATS).

As mentioned, the power of the Putnam-based model is augmented by other models, most notably those of Doty [ref. 7], Walston and Felix [ref. 24], and Halstead's book, Software Science.

The Doty model of cost estimation is programmed into the INCO cost model and is available through the option menu for use by the costing analyst. Experience with the Doty equations has produced very favorable results by way of convergence in calibration attempts with known cost data and the estimates of other cost models, namely the PRICE S cost estimation model. Subsequently, the Doty model was the primary choice for estimation purposes under the scaled systems research effort.

2.3.3 Current Stage of Development.

The current capabilities of the model are illustrated in Figure 2-10. As in "Step 3" of Figure 2-08, the model is still in the calibration and enhancement stages. This is perceived as an on-going phase since a software cost model is never really "done". The INCO model was designed with an eye for evolution and adaptability as more becomes known about the science of software cost estimating and as cost estimates can be traced through to their respective actual costs. Specifically, INCO is exploring credible, verifiable methods in which the technology constant can be more accurately determined. This has evolved to a set of

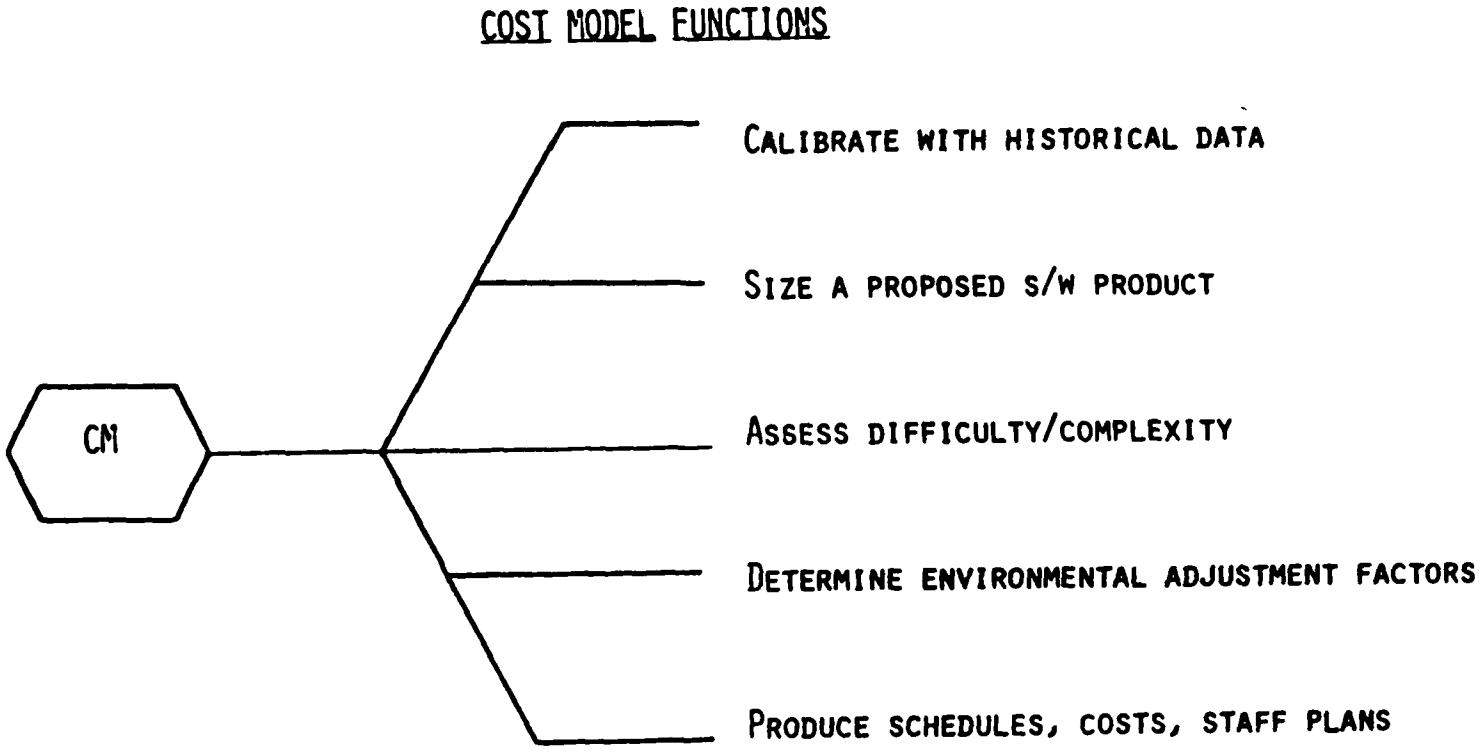


Figure 2-10 Capabilities of INCO Cost Model

adjustments based upon environmental, product, technological, and organizational factors. These adjusting factors have been aided by research such as that performed under the Scaled System project already mentioned. A brief enumeration of these factors is shown in Figure 2-11.

Keeping abreast of the current trends, the INCO model utilizes a modified version of Putnam's "software equation" - the same as that used by SAMSO's SPO [ref. 9].

With the trend toward better dissemination of information, particularly in the area of graphics, INCO has already begun the design and development of general-purpose graphics capabilities for its microprocessor-based hardware. With the ever-increasing advancements being made in the low-cost end of this hardware market, INCO has in sight the reality of truly cost-effective generalized graphics capabilities and hopes to enhance the cost model with such facilities.

Given the time and a few more advancements in the various technologies, INCO is confident in its ability to produce a true software life cycle cost and cost estimation model with a full complement of graphical capabilities and on low-cost hardware intended for in-house operation and ownership.

COST MODEL FACTORS

PRODUCT CHARACTERISTICS

- SIZE (DSLOC, DEMIs)
- TYPE/APPLICATION
- SPECIFICATIONS/PLATFORM
- PROCESSOR/SYSTEM LOADING
- INTERFACES

TECHNICAL CAPABILITY

- EXPERIENCE & SKILL LEVEL OF PERSONNEL
- PRODUCT FAMILIARITY
- DEVELOPMENT AIDS, FACILITIES

ENVIRONMENTAL CONSIDERATIONS

TECHNICAL

- FIRMNESS OF REQUIREMENTS SPECIFICATION
- FREQUENCY OF CHANGES TO REQUIREMENTS
- NEW, UNFAMILIAR HARDWARE
- CONCURRENT H/W & S/W DEVELOPMENT
- OFF-SITE DEVELOPMENT
- MULTI-SITE DEVELOPMENT
- REAL-TIME OPERATION
- DEV. SYS. DIFFERENT THAN TARGET SYS.
- INTERACTIVE VS. BATCH DEVELOPMENT
- PHYSICAL ACCESS LIMITATIONS
- CODE INVENTORY

MANAGERIAL

- SCHEDULE REQUIREMENTS
- AVAILABLE MANPOWER
- RESOURCE COORDINATION
- ECOs/SOPs
- LABOR RATES
- ECONOMICS/INFLATION RATES

Figure 2-11. Summary of Cost Estimation Factors

SECTION 3. SPECIFIC RESULTS

The scale factors and metrics that have been defined are described in Section 3.1, with further details to be found in the reports "Software Scale Parameters" and "System Scale Factor Metrics" (Appendices B and C). The interrelationships among scale factors derived as results of experimentation with the operating system performance simulator are discussed in Section 3.2. The basic and generalized decision factors and guidelines to be used by system architects in determining when scaled systems should be used as a part of a design effort are discussed in Section 3.3, and anticipated cost benefits of scaling are discussed in Section 3.4.

3.1 Scale Factors

Software scale parameters are those aspects of automated systems that can be reduced in scope in order to implement a cost-effective system scaled with respect to the full-scale system objectives. The development of a list of software scale parameters was accomplished in Task 1, Subtask 1, and described in the report "Software Scale Parameters" (Appendix B). The categories of software elements determined to be applicable to scaling were identified as data base, performance, functionality, security, maintainability, reliability, language, and hardware configuration. This section discusses aspects of system development that contribute to system cost and performance, and that are amenable to scaling.

3.1.1 Data Base

Data base characteristics include data base complexity (of access method and data structure) and data base size (number and length of

files, number of access keys, number and length of fields). Data base access complexity may be scaled by first employing the access method that would be the simplest for that size date base and then developing the scaling relationships involved in increasing the complexity, e.g., from sequential to indexed sequential to random access.

Some data bases deal with a relatively small set of different items. For example, the data base for an inventory control system might include only the following information: part number, description, quantity on hand, reorder point, supplier, reorder quantity, and unit cost. Most intelligence data bases, on the other hand, include a wide variety of information, covering such diverse subjects as different orders of battle, lines of communication, vessel movements, political and economic data, biographical information, etc. Data bases containing many different types of information are clearly more difficult to implement than are those limited to a very narrow subject area. As the diversity of a data base increases, development costs also increase due to the necessity to define additional data formats and structures, to possibly develop different data base load programs, and to probably implement new application programs.

The number of different data types does not, per se, have a significant impact on performance. As the number of different data types increases, there may be some additional overhead to search directories for control records for specific data types, but this overhead is usually insignificant compared to that required to locate a specific data item of a given data types. Hence, the major performance impact is associated with the volume of data, which might be expected to increase as the

number of different data types increases. The main reason for scaling the number of different data types relates to implementation cost. Restricting a scaled system to a subset of the total number of required data types may reduce the amount of data definition required, the variety of data base load programs necessary, and the number and complexity of application programs within the scaled system.

The amount of data resident within a data base, usually measured in terms of characters or records, tangentially impacts cost and significantly impacts performance. Neglecting all factors other than data volume, it should theoretically be just as simple to implement a large data base as a small one. A data base management system and the related application programs should be capable of handling any volume of data required by a system. However, performance impacts of data volume dictate that additional sophistication be implemented for processing large data bases than for small ones, in order to maintain an acceptable level of performance. For a small data base, therefore, a sequential file organization may be adequate. To achieve acceptable performance from a large data base system, however, a more complex data storage technique, such as a hierarchical or network structure, is usually required. The additional complexity required by additional data volume obviously adds to the cost of large data base systems.

While not direct software implementation costs, additional life cycle management costs are incurred by large data bases. The initial process of loading a large data base will cost more than that for a small one, due to additional data conversions, consistency corrections, and possibly manual entry required. Maintaining a large data base is also

more costly than maintaining a small one, due to the amount of checking that must be continually performed to establish and maintain the integrity of the data.

As mentioned above, the performance of a data base system can be expected to decrease as the volume of data increases. The amount of performance decrease is dependent on the sophistication of the data access techniques employed. For example, performance of sequential data bases will degrade significantly as data volume increases. On the other hand, performance of hierarchical data bases may not be perceptibly influenced by wide variations in data volume, provided that the types of requests made upon the data base follow the established hierarchy. Performance on requests that require searching of the entire data base or significant portions thereof, will degrade markedly with increases in data volume regardless of the data base structure employed. The major objective in scaling data base volume is to simplify the implementation of a data base system. Reducing the volume of data naturally simplifies loading a scaled data base. In addition, less sophisticated data storage techniques can be used with reduced amounts of data. In extrapolating ultimate system performance from scaled system performance, allowances must be made for any additional data access sophistication to be implemented, as well as for performance impacts of increased data volume. With additional data access sophistication included in the ultimate system, its performance may be equal to or better than that of a scaled system, even though the volume of data is dramatically increased.

Data base conceptual complexity is used here to denote the degree to which the data elements within a data base are mutually interdependent.

Conceptually simple data bases contain data which do not depend, to any great degree, on other data within the data base. For example, a data base used by a magazine publisher may include data on subscribers, advertisers, contributors, and production mechanics (ink and paper inventories, etc.). These four types of data bear no relation to each other. On the other hand, an intelligence data base might contain data on enemy weapon positions, technical weapon characteristics, friendly installation locations, and intelligence sources. Enemy weapon positions are correlated with technical weapon characteristics to determine their threat to friendly installation locations. All data is also correlated according to the intelligence sources. This is an example of a conceptually complex data base, with many types of information dependent on other types. A conceptually complex data base is far more expensive to implement than is a conceptually simple one. Data structures must be designed that permit rapid correlation of different types of information, and applications must be designed to maintain the integrity of all data interrelationships. Conceptually complex data bases will typically not perform as well as comparable conceptually simple ones. Extensive data correlations require additional data base accesses, as well as data base storage overhead to maintain efficiency.

Many data correlations can be unimplemented, implemented via manual means, or implemented through a semi-automatic technique such as multiple queries with intermediate hit files for a scaled system. This can significantly reduce the cost of implementing a scaled system. Relative performance of the scaled and ultimate systems would depend on many implementation factors. The cost of implementation complexity is

generally dependent on the underlying conceptual complexity of the data. For conceptually simple data bases, a complex implementation will generally be more expensive than a simple implementation. The reason for this is that a simple implementation would suffice to fit the data definition, and adding complexity tends to increase cost. (A complex implementation may be required, however, due to the performance considerations noted above, based on data volume.) For conceptually complex data bases, a simple implementation will generally be more expensive than a complex one. This is because all application programs, with a simple data structure, must be aware of the complexities of the data relationships. With a complex implementation, a sophisticated data base management system typically relieves the application programs from consideration of many of the conceptual complexities. Cost aspects are clearly dependent on the number of application programs required, the degree to which the data base management system can insulate the application programs from the conceptual complexities, and whether a data base management system can be used intact or must be specially developed or modified. A complex implementation of a data base will generally yield better performance than will a simple implementation. This is because direct access techniques (directories and hashing) improve data access times, and pointers or links between records speed the processing of data interrelationships. There is, however, a point beyond which additional implementation complexity becomes overkill for the underlying conceptual complexity and data volume. Past that point, the overhead required to maintain seldomly-used directories or links may begin to degrade performance. In any event, any implementation complexity must be

carefully designed to parallel the conceptual complexity, thus improving performance for the precise uses to which a data base will be put.

Since a scaled system need not support the conceptual complexity, data volume, or performance of an ultimate system, data base implementation complexity is very amenable to scaling. Using a simple implementation methodology will, in general, result in significant cost savings, provided that conceptual complexity is likewise scaled. Thus, a series of simple flat files, without complex data dependencies, might be used in a scaled system instead of a complex hierarchical or network structure. Estimating ultimate performance based on such a scaled system requires detailed analysis of the advantages gained by going to a more complex implementation philosophy.

Some forms of data lend themselves very readily to proven data base technology, whereas other, more exotic, data forms are still being investigated for efficient exploitation within a data base. For example, a data base of bank transactions contains well-defined data, constructed in accordance with fairly rigid formats, and subject to easily expressed validity checks. Becoming slightly more exotic, a data base of bibliographic information contains much English language text. Many such data bases have been constructed, but research is still underway on improving the effectiveness and efficiency of such data bases. At perhaps the most exotic extreme, several research programs within the intelligence community are currently examining ways of using data bases of digitized imagery. Such data bases would contain enormous volumes of data, and would require special algorithms to effectively distill information from the imagery data.

The expense of implementing a data base increases as the data within it deviates further and further from forms normally stored within conventional data bases. This is primarily due to two factors. First, conventional data usually lends itself to easily-defined structures, whereas efficient structures and even expected access criteria for unconventional data are usually difficult to define. Second, the algorithms for manipulating conventional data have been implemented many times and are well-understood, while the algorithms for manipulating unconventional data are often the subject of ongoing research and development. The net result of these two factors is that implementation of conventional data bases can proceed in a straightforward manner from design with little risk, whereas implementation of exotic data bases often includes many design changes and continual experimentation, with the attendant high cost and risk.

The structuredness of conventional data forms lends itself to efficient implementations of such data bases. As mentioned above, efficient structures and expected access modes are often not known for the more exotic forms of data. This naturally leads to difficulties in implementing good performance for data bases containing such data. Since the use of unconventional data forms greatly increases cost and reduces performance, omitting such data from a scaled system will certainly make it much easier to implement. However, one of the reasons for building a scaled version of a system requiring exotic data forms will usually be to prove the feasibility of processing such data. Hence, the conventionality of data forms would typically not be scaled, with economies of scaled system implementation realized elsewhere.

3.1.2 Performance Indices

The classes of quantitative performance indices identified for scaling are productivity, interactive responsiveness, utilization and operating system organization. Productivity is composed of the amount of work that can be physically accommodated and the rate at which it is ultimately accomplished. The amount of work can be measured by deriving the system capacity, the amount of information it can contain at any given period of time, as well as the capacity of the hardware components. The throughput, the average rate at which jobs are completed by the system in a given interval of time, is a result of nearly every aspect of a system configuration; from the hardware itself to the functions the system is required to perform to the typical set of jobs requiring system resources, i.e., the job mix. The scaled system design would have a scaled system capacity as well as a scaled job mix, structured for optimum performance. These factors all contribute to interactive responsiveness, the number of responses/unit time, the inverse of the time between the presentation of an input to the system and the appearance of the corresponding output. Because this parameter is difficult to predict on the front-end of the implementation phase, it will usually be quantified through observation. That is, a response time may be set as a target. The scaled system might reveal that the chosen design does not produce the required responsiveness. The full-scale system design specification could then be cost-effectively adjusted in the front-end of the design cycle, where economic leverage is the greatest.

Utilization is defined as the ratio of the time a specified part of the system is used to a given interval of time. Modules may be linearly scaled as the ratio between proposed and actual module utilization, where scale factors are in terms normal for the module, e.g., memory utilization is measured as a percentage of total memory available.

Operating system organization subelements were identified in "Software Scale Parameters" (Appendix B) as processing mode, operating system, and interrupt processing. These parameters represent a mode of operation rather than a measurable ratio and thus are difficult to quantify. However, the choice of one mode over another is a valid method to scale performance. Scaling system aspects applicable under this category would undoubtedly be highly case-dependent and quantifying the factors largely subjective.

3.1.3 Functionality

The approach to scaling functionality consists of reducing the variety of functions supported or reducing the functional complexity. The first method entails vertical functional scaling (eliminating subsystems); the second, horizontal functional scaling.

3.1.4 Security

Consider next the scaling of security functions. The degree of security provided for software and data is determined by the scope of access control, those attributes of software that restrict access to and manipulation of programs and data, and the completeness of access audit, the procedure whereby an historical record is maintained of both successful and unsuccessful attempts to access restricted data. Security may be considered a valid parameter for scaling when the scaled system

will be developmental in nature and when either adequate physical safeguards may be substituted for the full-scale software security procedures or the data to be protected is simulated or is non-sensitive public test data.

The basic goal of data base security is to prevent information from falling into the hands of individuals not authorized to receive it. Two major questions must be answered in designing a data base security system: How shall it be decided who has access to information, and what is the smallest unit of information to which access will be controlled?

The first question, that of determining individual access rights, has predominantly been answered through two different approaches, by user or by classification. The two approaches are sometimes also used together. The scheme controlling access by user effectively tags each item to which access is controlled with a list of those users allowed access to the item. Users requesting access to an item must be on the list for that item in order to be permitted access. The scheme controlling access by classification tags each item to which access is controlled with the item's security classification, special handling instructions, releasability, and so on. Each user, and perhaps terminal, has permission to access data with only certain security classifications, special handling instructions, and releasabilities. The system compares user access privileges with the classification of a requested data item before granting access.

The second question, that of the size of units of information to which access is controlled, has also been answered in several ways. Virtually all systems control access at the system level, with user

sign-on password authentication. Most systems control access to individual files in some fashion, and many systems even control access to individual records within files. Some systems go so far as controlling access to individual fields within records.

Related to security is the requirement to maintain an audit trail of all operations taken against the data. This audit trail normally contains more information than the transaction log maintained by a data base management system to support data integrity. Preserving data integrity requires logging of only update transactions, whereas a security audit trail also requires recoding of all data read from a data base as well. The degree to which security audit trails are implemented for typical intelligence systems varies. Virtually all systems record user sign-on and sign-off. Many systems also record major function invocation. Almost no systems record the actual data manipulated by users. Other aspects of system security include accreditation for operation with classified information and the problems of obtaining cleared programmers and facilities.

A security system can be considered scaled if it encompasses a file protection methodology less restrictive than the full-scale system. This scaling can take the form of, for example, a less sophisticated level of file protection, a smaller access matrix, elimination of codewords, audit trails, encryption, and/or simplification of the authentication mechanism.

3.1.5 Maintainability

Maintainability is defined as the probability that, when maintenance action is initiated under stated conditions, a failed system will be

restored to an operable condition within a specified time. It also refers to the effort required to locate and fix an error in an operational program and is a technically valid area for scaling, since the implementation of maintainability involves increased software development cost and/or time. Maintainability is a function of the capabilities included in the system, the skill level of the personnel, and the support facilities (locally available tools and diagnostic test equipment or aids, spare parts or alternative program versions or back-up files). Since scaling of this parameter would involve the elimination or simplification of functional requirements of the system, the approach would be similar to that for scaling functionality. However, eliminating modules whose purpose is to enhance maintainability may indeed prolong rather than enhance the progress of the project. Such considerations must be emphasized when scaling is contemplated.

Among the maintenance modules which could be scaled are process error handling (minimize the number of conditions to be checked), restart/recovery procedures, data correction, fault detection/trap software, monitors of system performance, and back-up procedures. Development and diagnostic aids such as program tracers and interactive debuggers might actually be added, to reduce the development effort of the full-scale system.

3.1.6 Reliability

Reliability can be defined as the probability of satisfactory performance for a given time when used under stated conditions, the metric being defined as the number of failures/time. A software failure is an occurrence of a software error, when the software does not do what

the user reasonably expects it to do. In order to prevent failures from occurring in the first place, a certain amount of redundancy is built into systems such that automatic diagnosis and recovery can be accomplished by the software itself without operator attention or intervention. This redundancy requires additional design, system storage, programming, and effort; thus reliability may be scaled with respect to these aspects.

Some reliability elements amenable to scaling would include precision, error detection software (eliminate software geared to errors which would occur infrequently in practice or not at all in the input to the scaled system), approximation algorithms (use fast, easy, not as accurate as possible approximation functions and algorithms), and coding standards. Relaxation in enforcement of coding standards might only be considered where recoding would be necessary to implement the full system. If, however, the scaled system will form the basic structure for the full system, then strict coding standards should be maintained.

3.1.7 Programming Language

Two aspects of programming language suitable for scaling are language selection and implementation. A language that is optimal for the scaled system but different from the one chosen for the full-scale system might be selected if the target system is to be completely recoded. Such a language might be chosen based upon considerations of top-down design, code readability, and modifiability, thereby contributing to accelerated program development. Scaling language implementation could be accomplished by successively enhancing a baseline subset of the language being implemented.

3.1.8 Hardware Configuration

The choice of individual hardware components and their configuration is an important aspect of the scaled systems methodology. Significant savings in schedule, effort and cost may be achieved by reconfiguring the target system hardware or by selecting an alternate operational environment for the scaled systems effort.

In order to reduce complexity, either the number of component types or the total number of components may be scaled, both approaches reducing total system complexity. Factoring hardware configuration is simplified by the nature of the entity itself, due to the numerically descriptive nature of hardware. Some hardware elements that it might be possible to scale are: number of CPU's (scale from multiprocessing to a single processor), number and/or type of peripherals, size of the instruction set of a CPU, input devices (simulate the data instead), number of communications nodes, complexity of communications network or hierarchy (lower the number of linkages among nodes), and level of service to peripherals (eliminate prioritized service).

3.1.9 Simulator Variable - Scale Factor Relationships

The scale factors introduced in the report "System Scale Factor Metrics" (Appendix C) influence and interrelate with each other in complex ways that can be quite different for different operating regimes (e.g., disk-limited, CPU-limited) of the IDHS being modeled. In order to scale a system, ways are needed of predicting changes in system characteristics as the scaled parameters vary, even when the variations are large enough to place the IDHS into a different operating region. For example, if the scaled system has a factor of four fewer terminals

than the envisioned full-scale system, it is necessary for the system designer to know how system throughput will degrade when the system is scaled up and terminals are added. The report "Interrelationships Among Scaling Factors" (Appendix D) addresses these considerations.

Figure 3-01 illustrates the throughput and CPU speed functional relationship, demonstrating that scaling a given system parameter will not necessarily affect another parameter in the same way in all operating regions; i.e., increasing CPU speed in a CPU-limited regime will affect throughput significantly, while in a disk-limited region, it will have very little effect. It can be said then that the relationships between parameters are complex and non-linear. It is not possible to write down analytic expressions that will hold under all conditions.

In order to provide the system designer with the tools that will enable him to predict performance under the wide-range of scaling conditions that are encountered in practical situations, a concept was evolved that uses a simulation model of a generalized IDHS to predict performance and to predict changes in one variable from changes in another; i.e., the simulation substitutes for the nonexistence of precise analytic functional relationships between various scaled parameters.

It has been found that a fairly small number of parameters is adequate to specify each particular IDHS to the simulation. Each of the input parameters, in turn, can be expressed as a fairly simple analytic function of the scaling parameter factors. A series of formulae are used in steps to relate the simulator variables to scale factors. A diagram of the technique is shown in Figure 3-02.

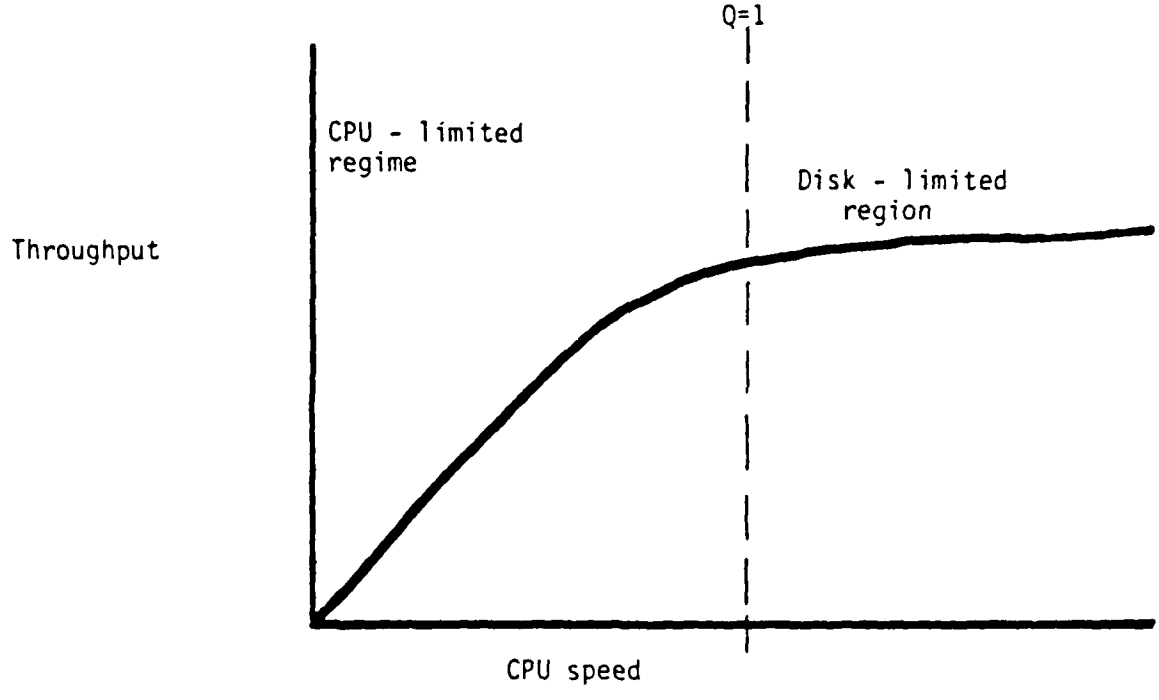


Figure 3-01. Throughput - CPU speed functional relationship

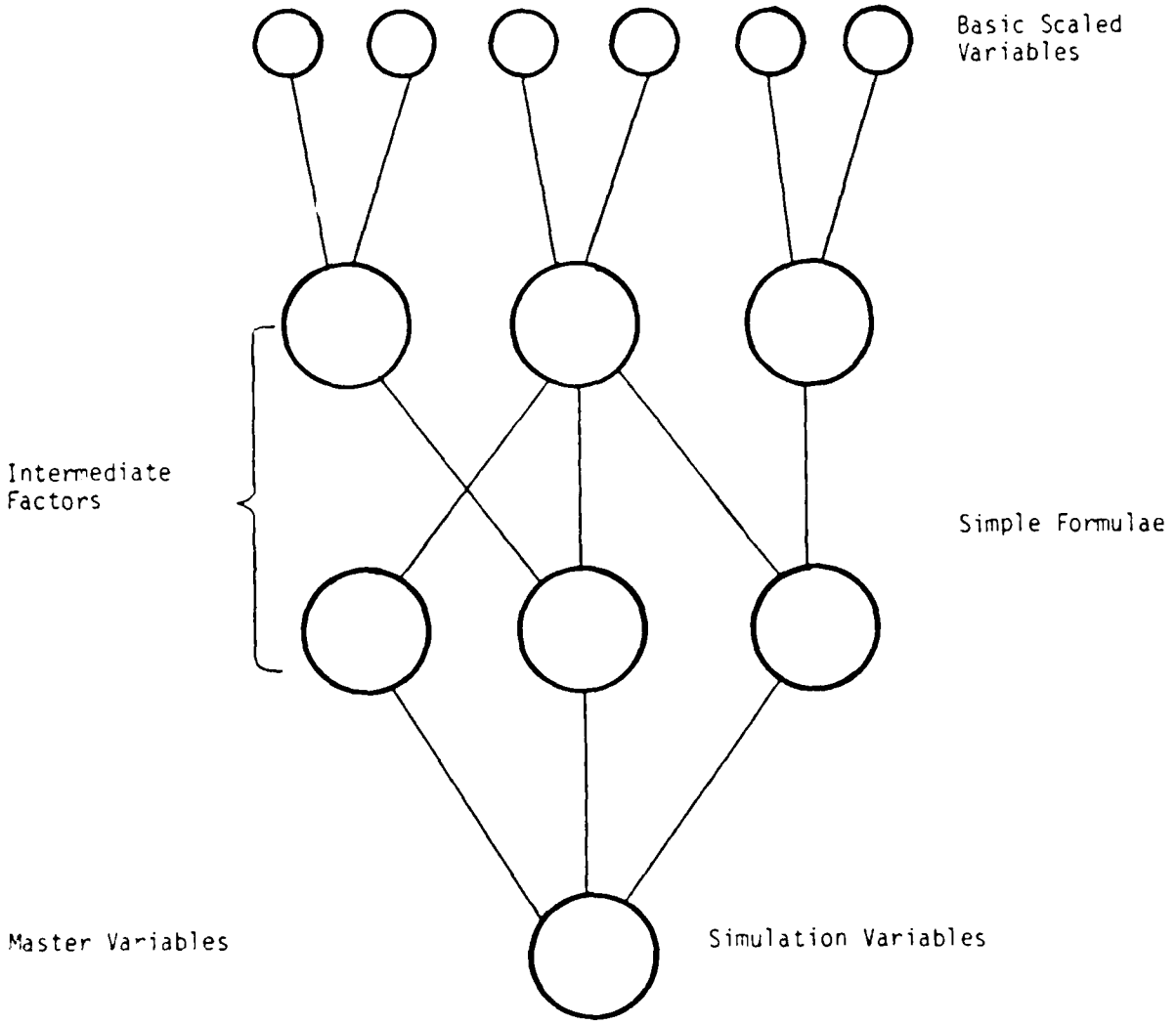


Figure 3-02. Relating Scale Factors to Simulation Variables

As shown in Figure 3-03, an example of a simulator input variable is CPU service time, i.e., time in CPU per CPU block, where a block is a set of instructions until a disk access is encountered. The following formulae, as described in the report "Simulator Variable-Scale Factor Equations" (Appendix D), are one set that can be used to relate CPU service time to system scale factors.

CMEAN can be defined as follows:

$$\text{CMEAN} = \frac{\text{instructions executed/block}}{\text{instruction/time (power)}}$$

Define the following terms:

N_D = number of disk accesses

= N_{DB} (number of data base accesses) + N_{DP} (number of paging disk accesses)

I = number of instructions

= I_C (number of computational instructions) +

I_{DB} (number of data base instructions) + Ip (number of paging instructions)

Define the frequency of data base disk accesses per computational instruction,

$$F_{DB} = \frac{N_{DB}}{I_C}$$

Then $N_{DB} = \frac{I_C \ [N_{DB}]}{I_C} = I_C \ F_{DB}$

Also $N_{DP} = K_p * I_C * \frac{C_V}{C_R}$, where K_p is a system-dependent constant calculated as the number of paging accesses/computational instruction, C_V is the virtual core for a particular job (the job size) and C_R is the real core for the job (the

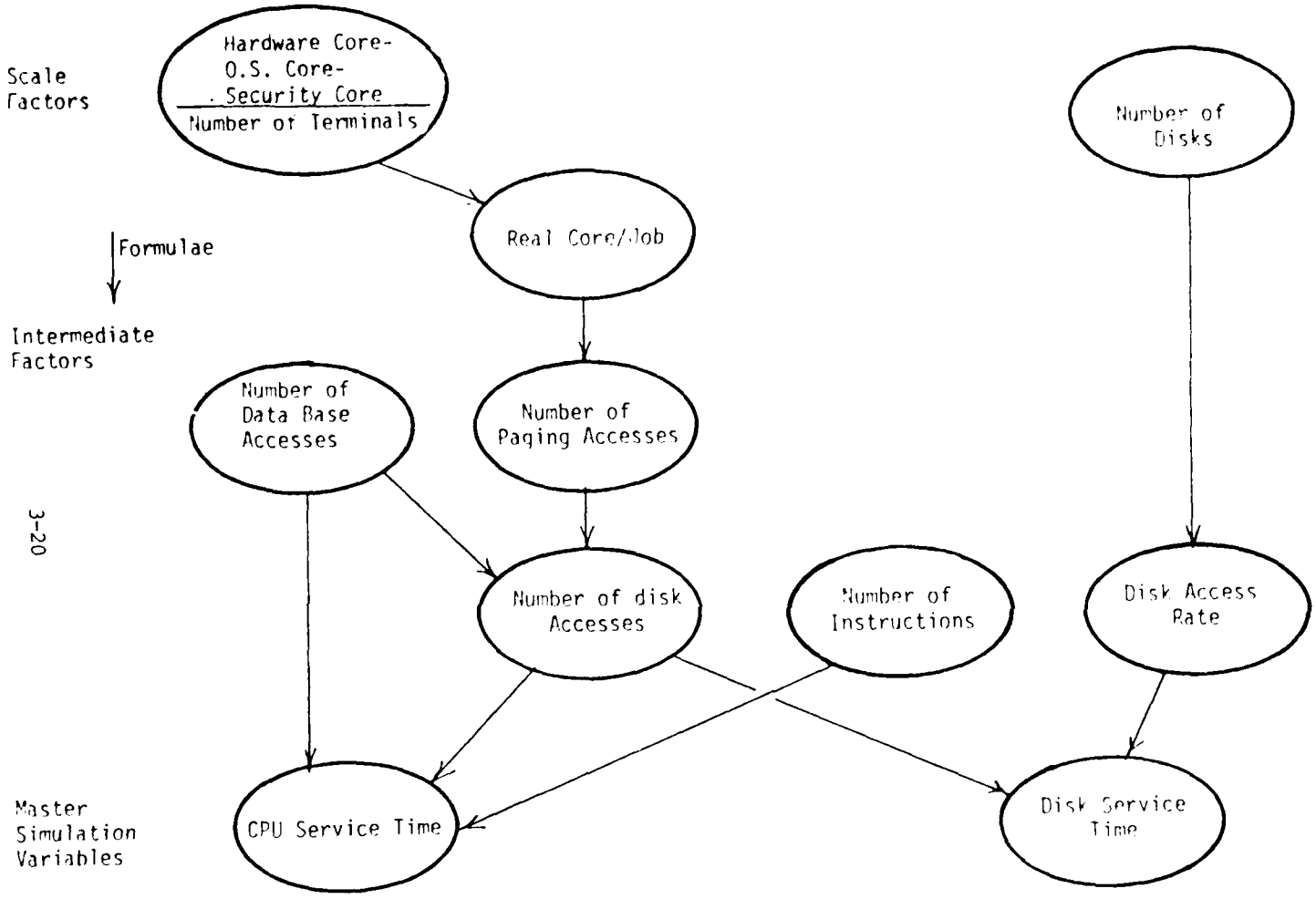


Figure 3-03. Relating Scale Factors to CPU and Disk Service Time

actual core available for the job).

Then

$$N_D = N_{DB} + N_{DP}$$

$$N_D = I_C \left[F_{DB} + K_P \frac{C_V}{C_R}\right]$$

$$\frac{I_C}{N_D} = \frac{1}{F_{DB} + K_P \frac{C_V}{C_R}}$$

I_C is the number of instructions per block so,

$$\frac{N_D}{C_{MEAN}} = \frac{\left[F_{DB} + K_P \frac{C_V}{C_R}\right] - 1}{\text{instructions/time}}$$

To find a value for F_{DB}, estimates and typical numbers will be sought. The value will depend on the function being performed and the probability of having to make a disk access. There are several factors that affect the probability that a piece of information is in core vs. on disk, such as the amount of the data base that is stored in core at any time, the organization of data on the disk (the data base structure), and the data manipulation algorithms. Also, since F_{DB} was defined as $\frac{N_{DB}}{I_C}$, it may be possible to calculate N_{DB} for a given function and data base organization, while I_C would also be a function of scale factors, such as the function being performed and the data base size and data base complexity. Thus F_{DB} could be derived in this way.

A way to approach the problem of evaluation might be to start with "reasonable" estimates for these parameters. When the scaled system is operational, they can be measured by monitoring system behavior. Indeed, the purpose of building the scaled system is to measure the parameters which will be used in the full-scale system so that flexibility in the

design of the full-scale system can be retained. The small scale system together with the simulation will enable the designer to see what will work in the full-scale system.

To find C_V for each job of type j, assume input values for simulation parameters mean $C_V(j)$, $\sigma C_V(j)$. As the job begins, pick the actual C_V according to a probability distribution function.

For C_R, the real available core for the job, the following system-dependent values can be input:

C_T = total core for the machine

C_{OS} = operating system core

Then

$$C_R = \frac{C_T - C_{OS}}{\text{number of jobs running}} = \frac{C_T - C_{OS}}{\text{number of terminals.}}$$

In the equations that have been discussed, the simulator parameters have been defined as functions of many of the scale factors, including power (instructions/time), number of terminals, real core (system capacity), number of instructions (related to data base complexity and structure), hardware, functionality, and security core (involved in the calculation of C_R, the real available core for a job). The use of the simulation then permits analysis of scale factor interrelationships.

The simulation, as described in the report "Simulator Description" (Appendix E), will be used by the system designer in an iterative manner in the course of specifying the full-scale system. The scaling factors will be specified and used as input to the simulation, the output will be examined, and scaling will be respecified until the desired outputs, i.e., full-scale system behavior, are achieved. A typical question would

be: How much can the data base size be scaled up with present disk hardware without going below the minimum required responsiveness (responses per unit time)? Will it be necessary to have more and/or faster disks in order to achieve the desired full-scale system responsiveness and incorporate the necessary data base size? If access time is improved by so much, how much can the data base then be scaled up? The system designer will look at the results of the simulation based on a set of values for the scaling parameters and iteratively adjust these values. Such respecifications of scaling may well result in design changes for the full-scale system, e.g., by going to more and/or more powerful hardware. Thus the tools to be used will be the simulator and the set of input variables.

3.2 System Design Methodology for Using Scaled Systems

3.2.1 Interrelationships Among Scaling Factors

The objective of this task was to develop standard procedures for applying the scaled system technique to new IDHS development projects and to describe how measurements made on a scaled system can be extrapolated into predictions for a full-scale system.

The result of Task 1 was a set of scale factors that describe an IDHS, with appropriate metrics defined on them. The goal of Task 2 was to determine the predictive value of each scaled parameter before preparing guidelines for which parameters to scale. Toward this end, an operating system performance simulator was designed, as discussed in the report, "Simulator Description" (Appendix F).

A job enters the system at random intervals chosen from a Poisson distribution, from one of n terminals ($n \leq 100$). The exponential

probability distribution function is used to model the job arrivals because, as noted in Beizer [ref. 6], assuming this distribution is equivalent to saying that the arriving customers individually and collectively behave as if they were not aware of each other's existence, because it is usually (but not always) pessimistic, and because it leads to reasonable expressions for the queueing parameters. The use of the exponential interarrival time distribution leads to a Poisson arrival rate distribution.

The job is assigned a job class (CPU- or disk-bound) and a CPU/disk iteration count, based on probability distributions. Each terminal is assigned a wait or "think" time. The job is placed on the CPU or disk queue if the required facility is busy; when it gains control of the CPU, it is assigned a CPU service time and a disk service time.

Experiments with sets of various parameter values were performed in order to address the issue of how such factors as the number of terminals, the job mix (the combination of CPU-bound and disk-bound jobs), and average CPU service time affect disk waiting time, CPU and disk utilization, response time, and other measures of system performance. Tests were run with 8, 10, 16, 20, 50, 70, and 100 terminals, with the percentage of CPU-bound jobs ranging from 10% to 90%. Figure 3-04 summarizes the experimental results for tests involving mean CPU/disk iteration count (the number of times the job flips between the CPU and the disk) of from 4 to 10 for CPU-bound jobs and 12 to 30 for disk-bound jobs.

It is necessary to examine the parameter interrelationships and system behavior in different operating regions, i.e., CPU- or disk-bound.

% CPU-Bound	No. of Terminals	Response Time		
		$\mu_c = 4, \mu_d = 12$	$\mu_c = 8, \mu_d = 24$	$\mu_c = 10, \mu_d = 30$
10%	8			
	10			
	16			
	20			9898
	50			22001
	75			32851
	100			46594
15%	8			
	10			4731
	16			
	20	4046	6961	9878
	50	9050	17084	20511
	75			33221
	100	16039	34059	45173
25%	8	1378	2692	
	10	1888	3525	4445
	16	2986	6130	
	20	3849	6847	8592
	50	7571	15119	18943
	75			28073
	100	15483	30893	31654
50%	8	1219	2324	
	10	2347	2764	3651
	16	2041	4009	
	20	2788	4234	6680
	50	6380	11288	14238
	75	8802	14930	17009
	100	12296	19706	27399
90%	20			3739
	75			13215

Figure 3-04 Summary of Simulator Experimental Results

Figure 3-05 plots the number of terminals against the response time (the inverse of the responsiveness scale factor) for a system that has approximately 15% of its jobs in the CPU-bound category. The graph shows that for a mean of 4 CPU/disk iterations for CPU-bound jobs and a mean of 12 CPU/disk iterations for disk-bound jobs, the increase in response time is close to being proportional to the increase in the number of terminals, i.e., increasing the number of terminals by a factor of 2.5 (20 to 50) results in a 2.2 fold increase in the response time, while doubling the number of terminals from 50 to 100 results in a factor increase of approximately 1.8 in the response time. As the mean number of CPU/disk iterations increases, the increases in response time for the higher number of terminals are sharper, as can be seen from Figure 3-05.

Figure 3-06 shows the number of terminals plotted against the response time in the operating region where 25% of the jobs are CPU-bound. The curve for μ_c=4, μ_d=12, follows the same pattern as the 15% CPU-bound curve in Figure 3-05; i.e., the sharp increases in response time take place when the mean number of CPU/disk iterations is highest.

Similar phenomena are demonstrated in Figure 3-07 in the region where 50% of the jobs are CPU-bound and in Figure 3-08 where 10% and 15% of the jobs are CPU-bound. The general conclusion illustrated by the results of these experiments is that response time increases as the number of terminals increases, with proportionately larger increases taking place at the higher range of number of terminals and in the regions where more jobs are disk-bound; i.e., the curves tend to flatten as the percentage of CPU-bound jobs increases. In addition, as might be expected, response time increases as the mean number of CPU/disk

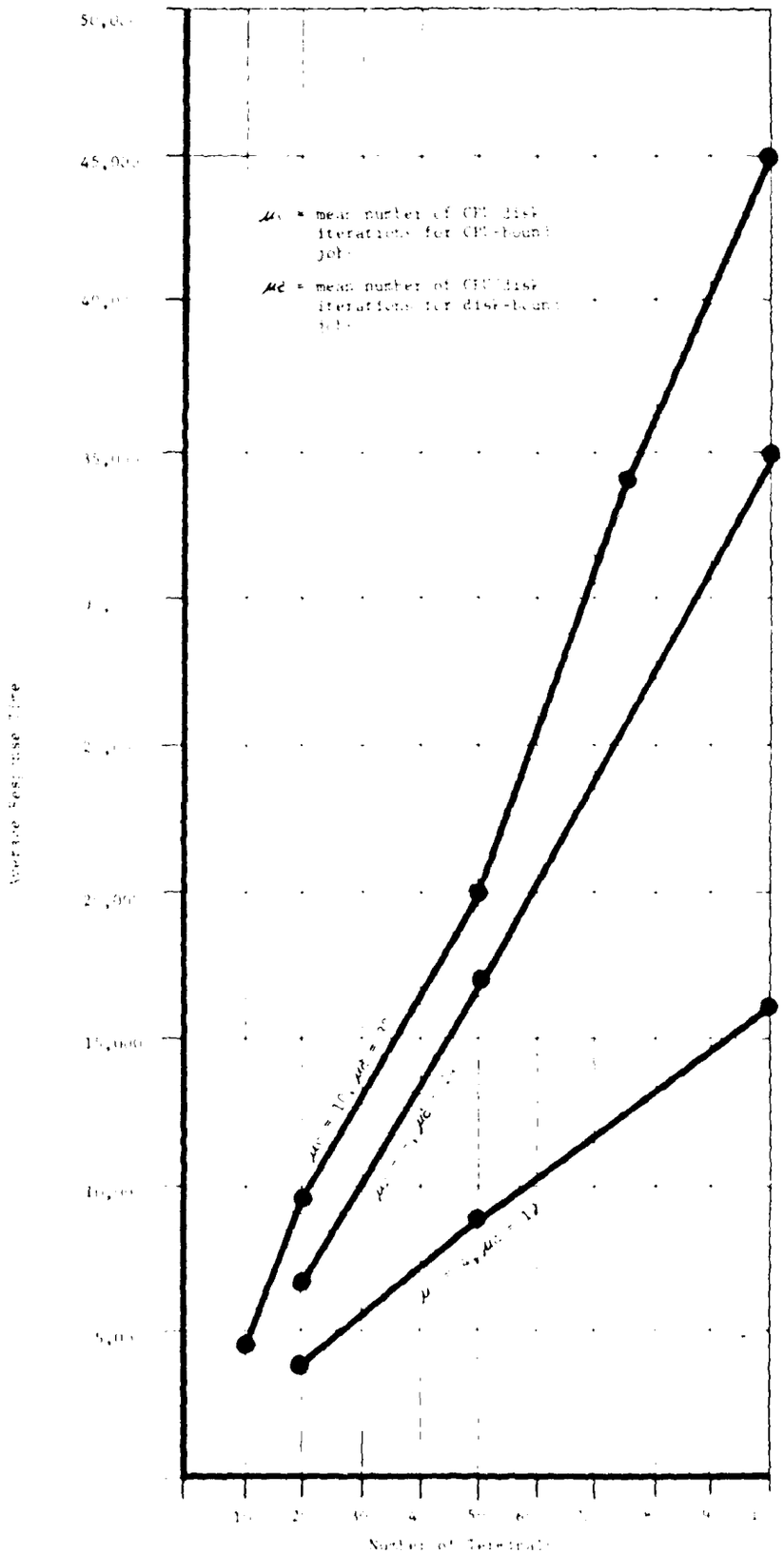


Figure 3-05. Number of Terminals vs. Average Response Time with 15% CPU-bound jobs

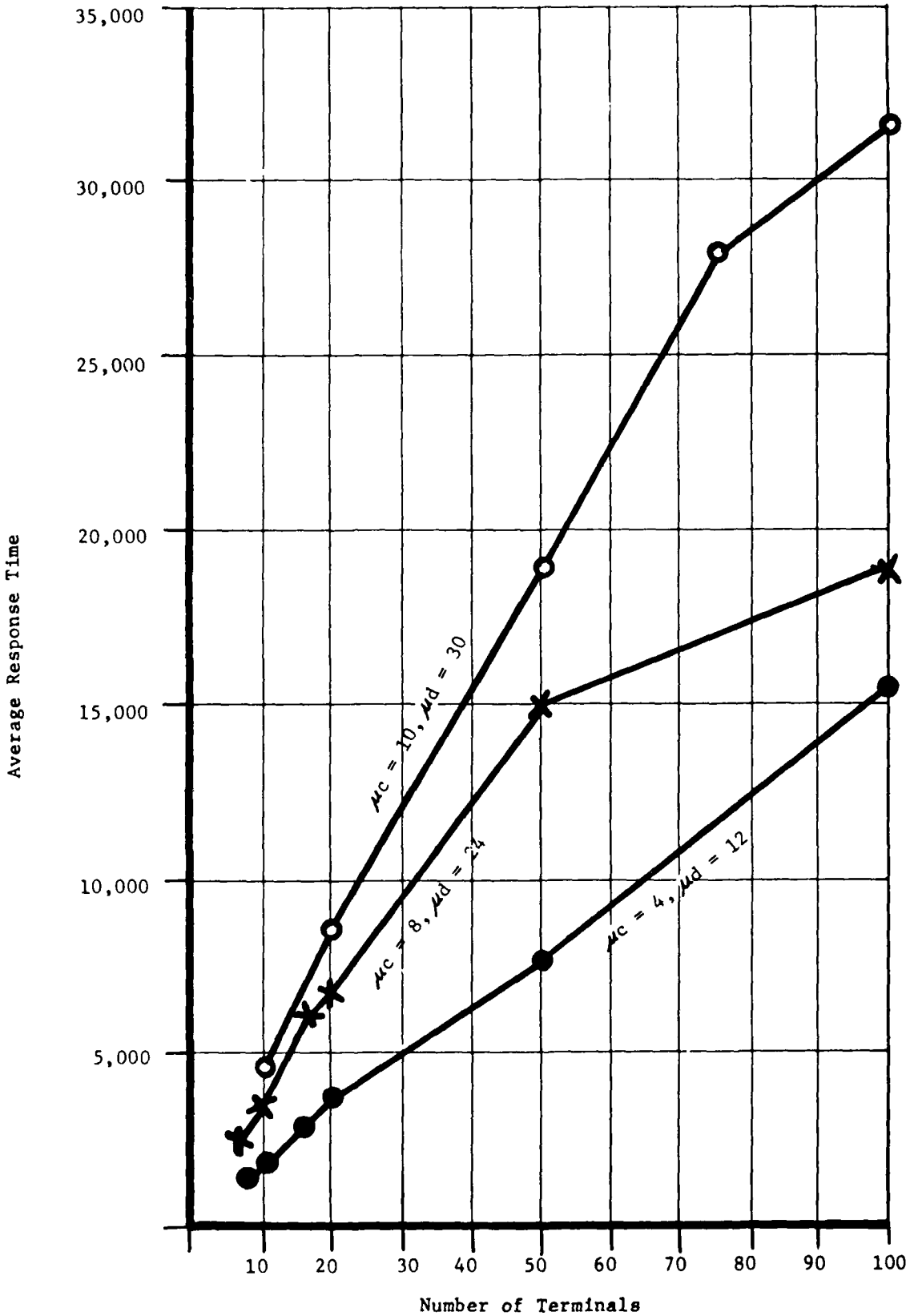


Figure 3-06. Number of Terminals Vs. Average Response Time with 25% CPU-bound Jobs

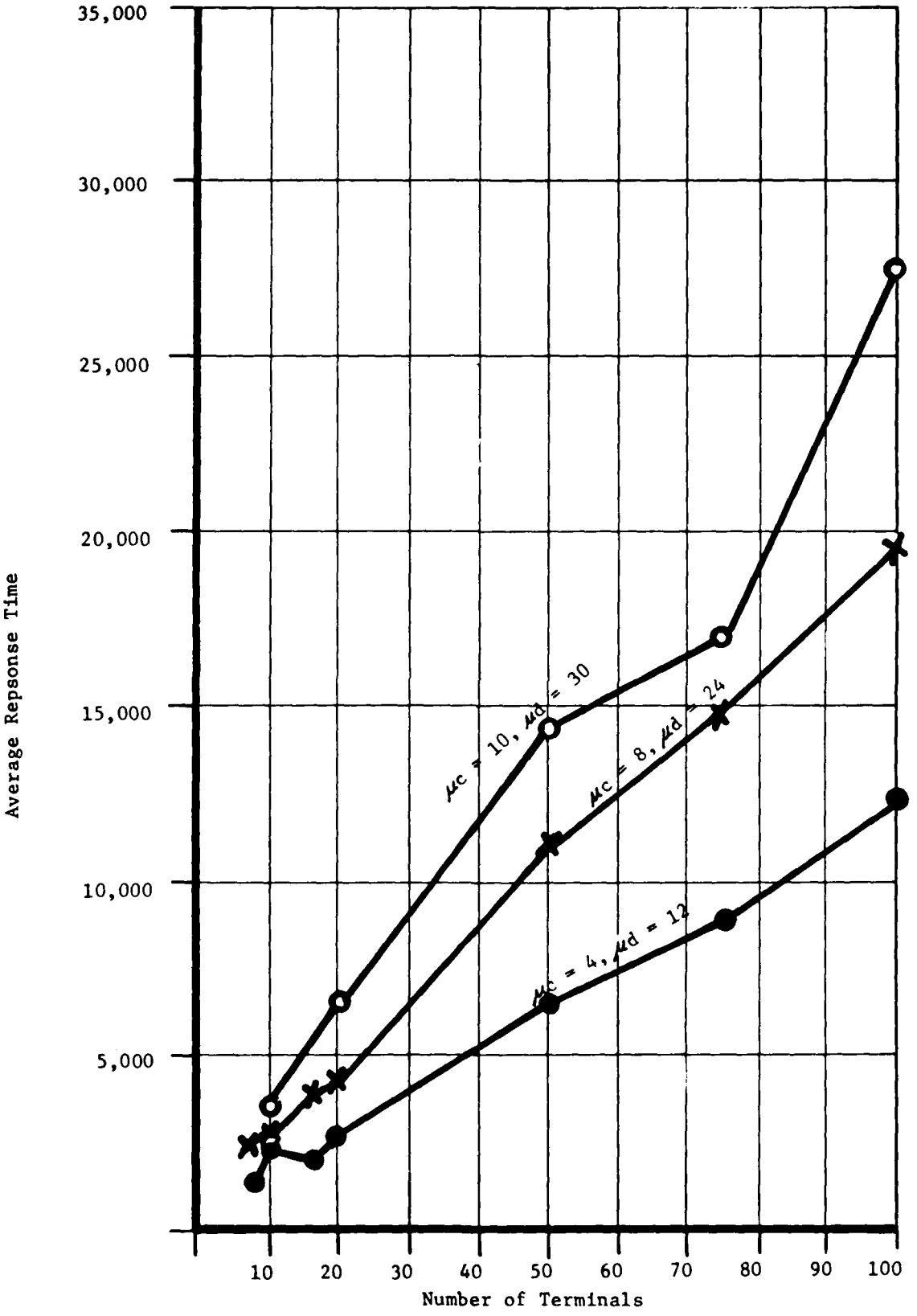


Figure 3-07. Number of Terminals vs. Average Response Time with 50% CPU-bound Jobs

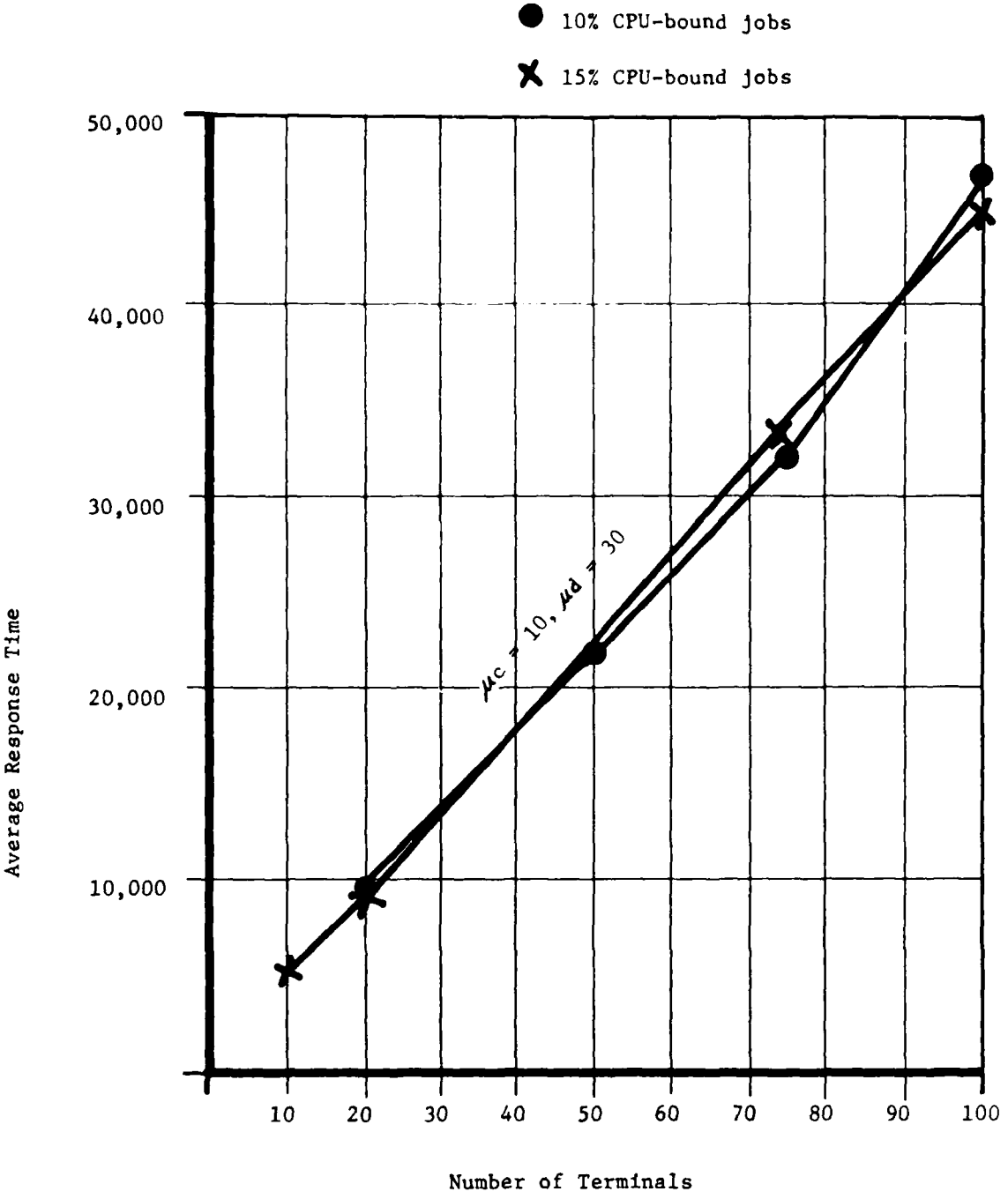


Figure 3-08. Number of Terminals vs. Average Response Time with 10% and 15% CPU-bound Jobs

iterations increases. The rate of increase is not predictable however. Figure 3-09 summarizes the changes in response times as the percentage of CPU-bound jobs decreases, i.e., the system becomes more disk-bound. It can be seen that doubling the number of CPU/disk iterations does not consistently double the response time, and the increases in response time vary within operating regions. Figure 3-10 summarizes the performance curves in Figures 3-05, 3-06, and 3-07.

Figure 3-11 demonstrates the system's behavior in the region where 25% of the jobs are CPU-bound and the number of CPU/disk iterations for CPU-bound and disk-bound jobs is 16, 32, or 40. Again, sharper increases are seen in the curves representing 50 and 100 terminals, as compared with those curves for 8 to 20 terminals. Figure 3-12 illustrates similar behavior for an environment where 50% of the jobs are CPU-bound. Thus, in general, it can be said that adding CPU/disk cycles to the average job results in increased response time. Similarly, as demonstrated in Figure 3-13, the increase in the average disk wait time is approximately proportional to the increase in the number of terminals in all operating regions examined.

Figures 3-14 and 3-15 show what happens to the response time as the percentage of CPU-bound jobs increases. Generally, the response time decreases, with the sharper changes taking place for the curves representing the larger number of terminals. Figure 3-14 illustrates the results of the experiments with mean CPU/disk iteration count of 4 for the CPU-bound jobs and 12 for the disk-bound jobs (indicated as (4,12)) and those with mean CPU/disk iteration counts of 8 and 24, respectively. Figure 3-15 plots the percentage CPU-bound jobs vs. response time curve

No. of Terminals	(4,12) 50%:25%	(8,24) 50%:25%	(10,30) 50%:25%	(4,12) 50%:15%	(8,24) 50%:15%	(10,30) 50%:15%
8	+13%	+16%				
10	-24%	+28%	+22%			+30%
16	+44%	+53%				
20	+38%	+62%	+29%	+45%	+65%	+48%
50	+19%	+34%	+33%	+42%	+51%	+44%
75			+65%			+95%
100	+26%	+57%	+16%	+30%	+73%	+65%

Figure 3-09 Summary of Changes in Response Time

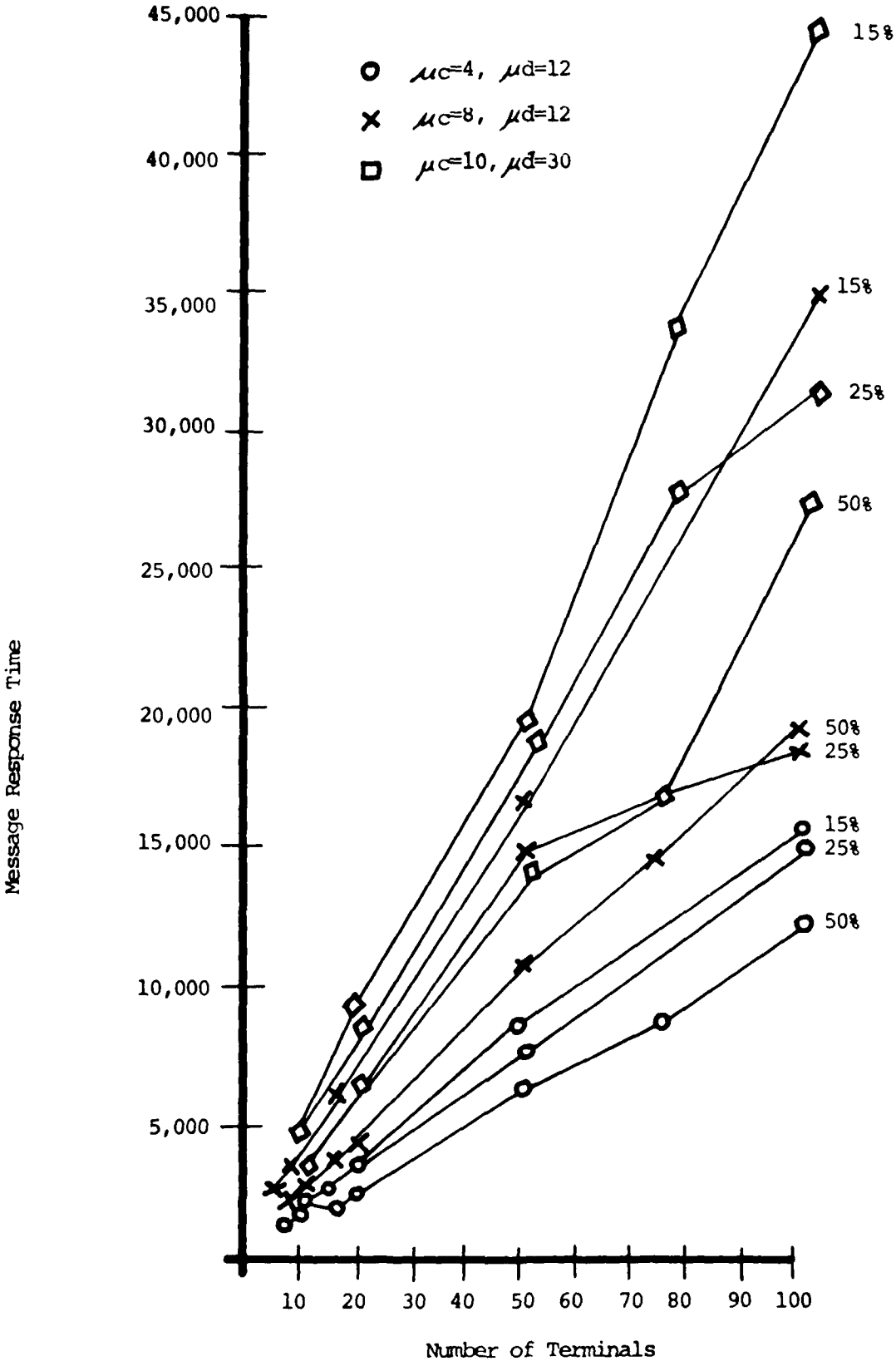


Figure 3-10. Summary of Performance Plots

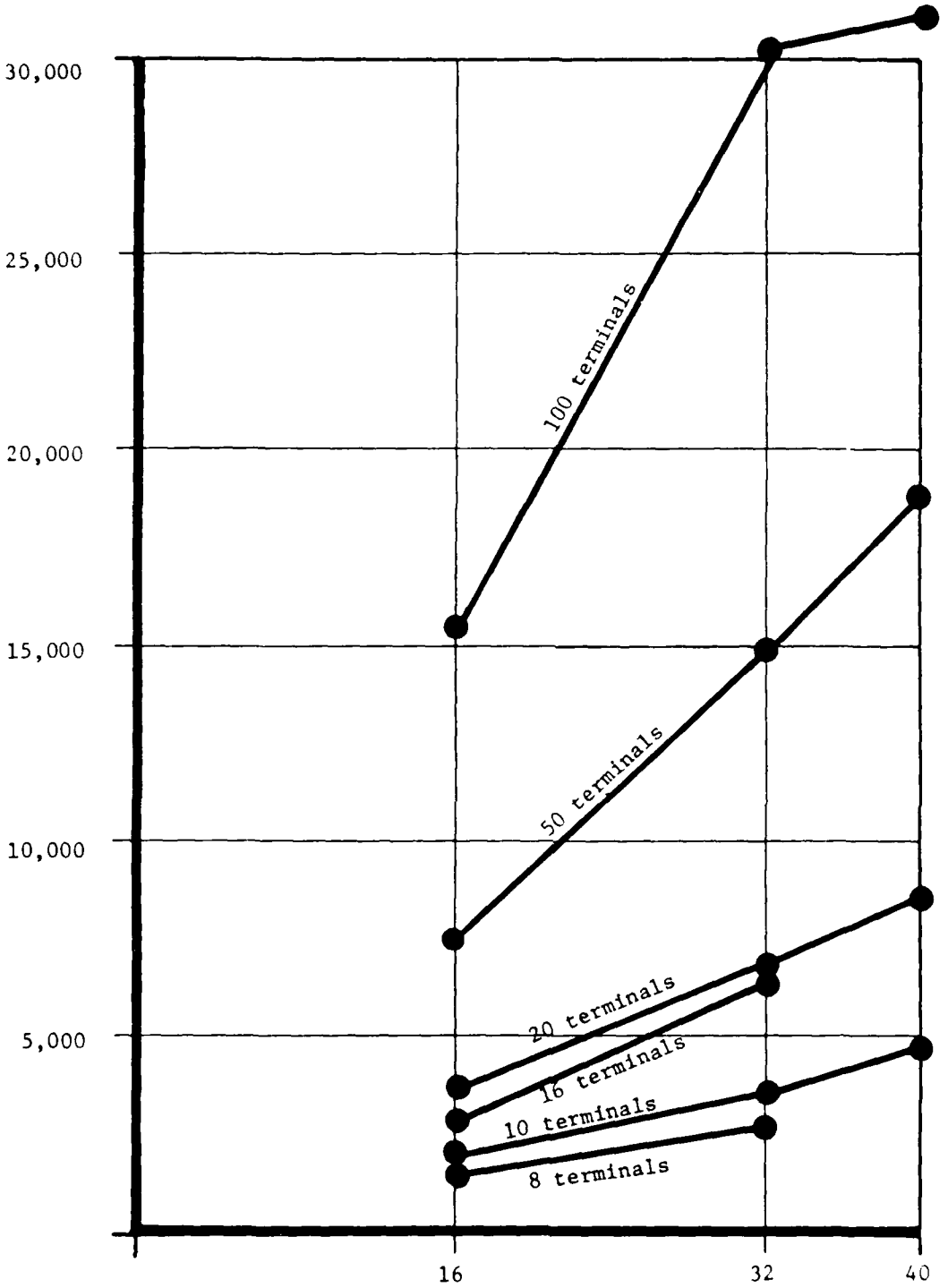


Figure 3-11. CPU/Disk Iterations vs. Average Response Time with 25% CPU-bound Jobs

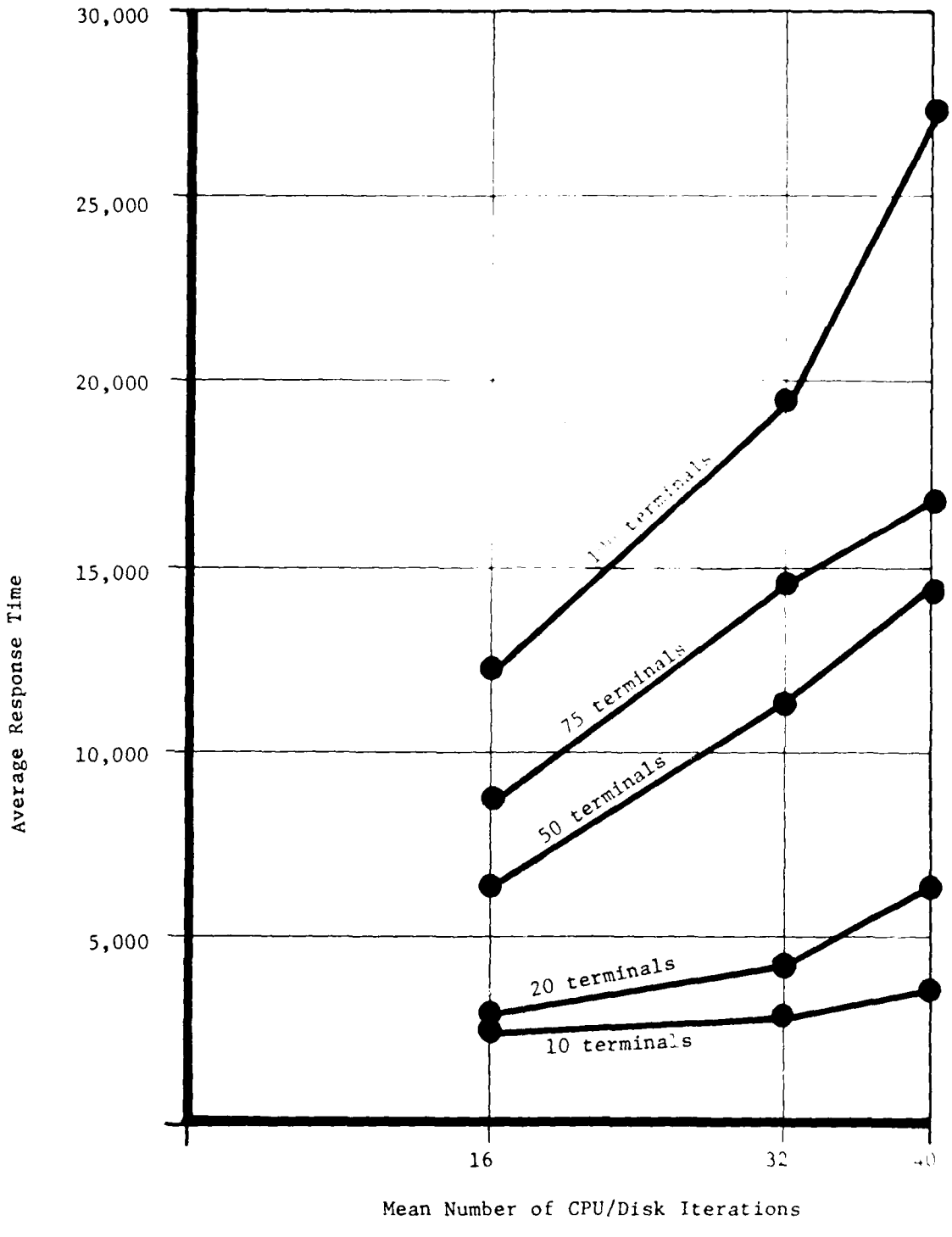


Figure 3-12. CPU/Disk Iterations vs. Average Response Time with 50% CPU-bound Jobs

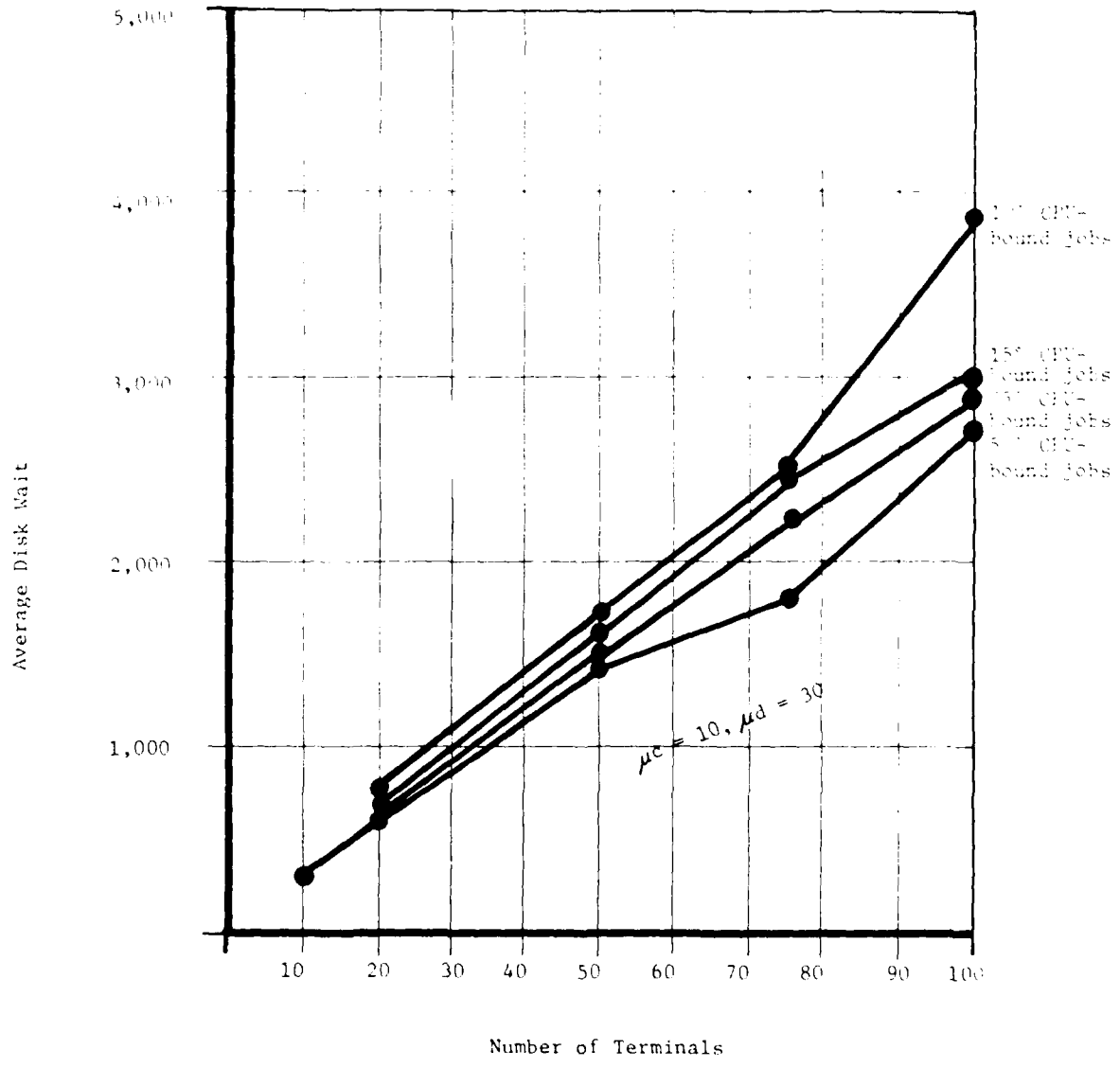


Figure 3-13. Number of Terminals vs. Average Disk Wait

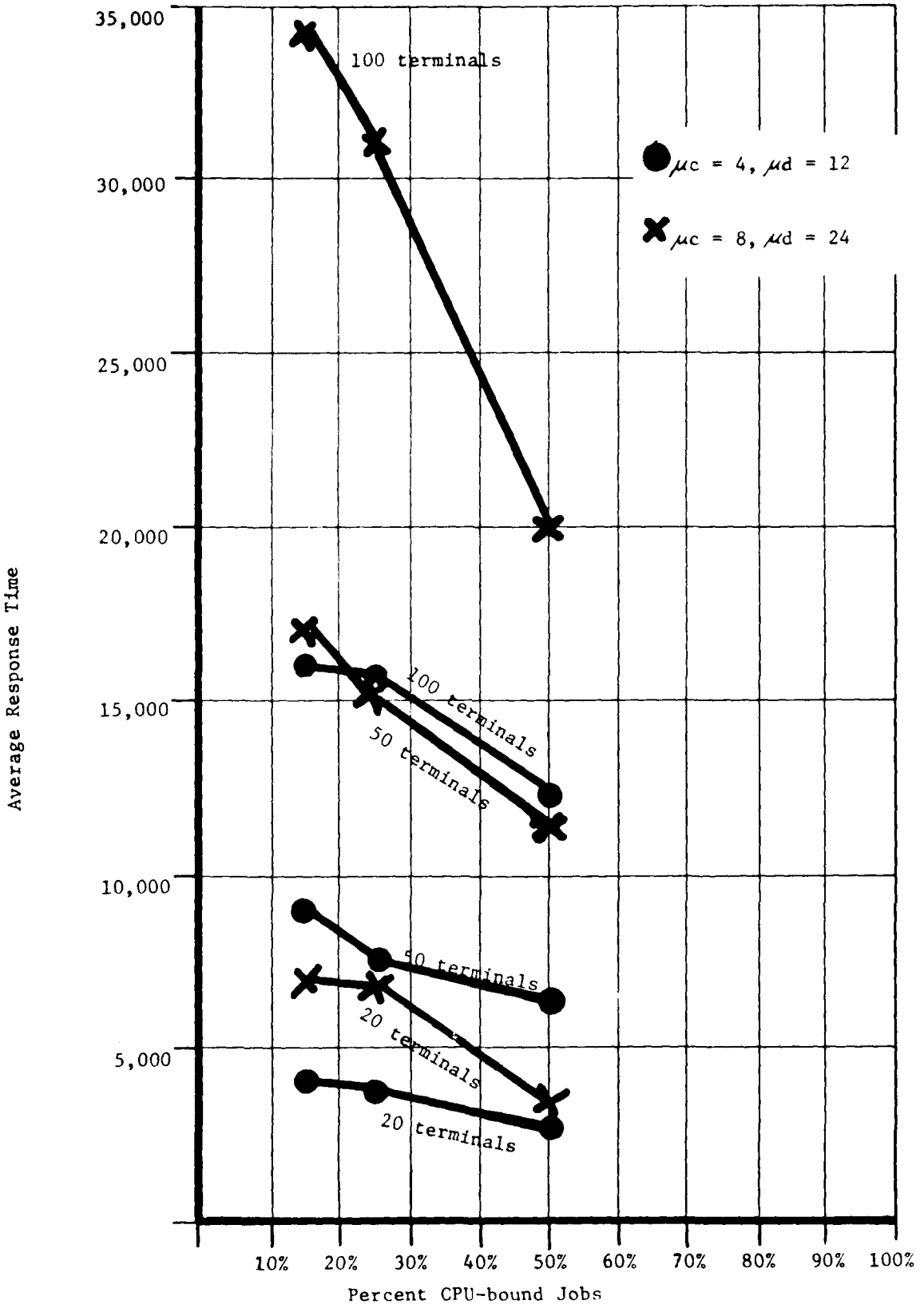


Figure 3-14. Percent CPU-bound Jobs vs. Average Response Time

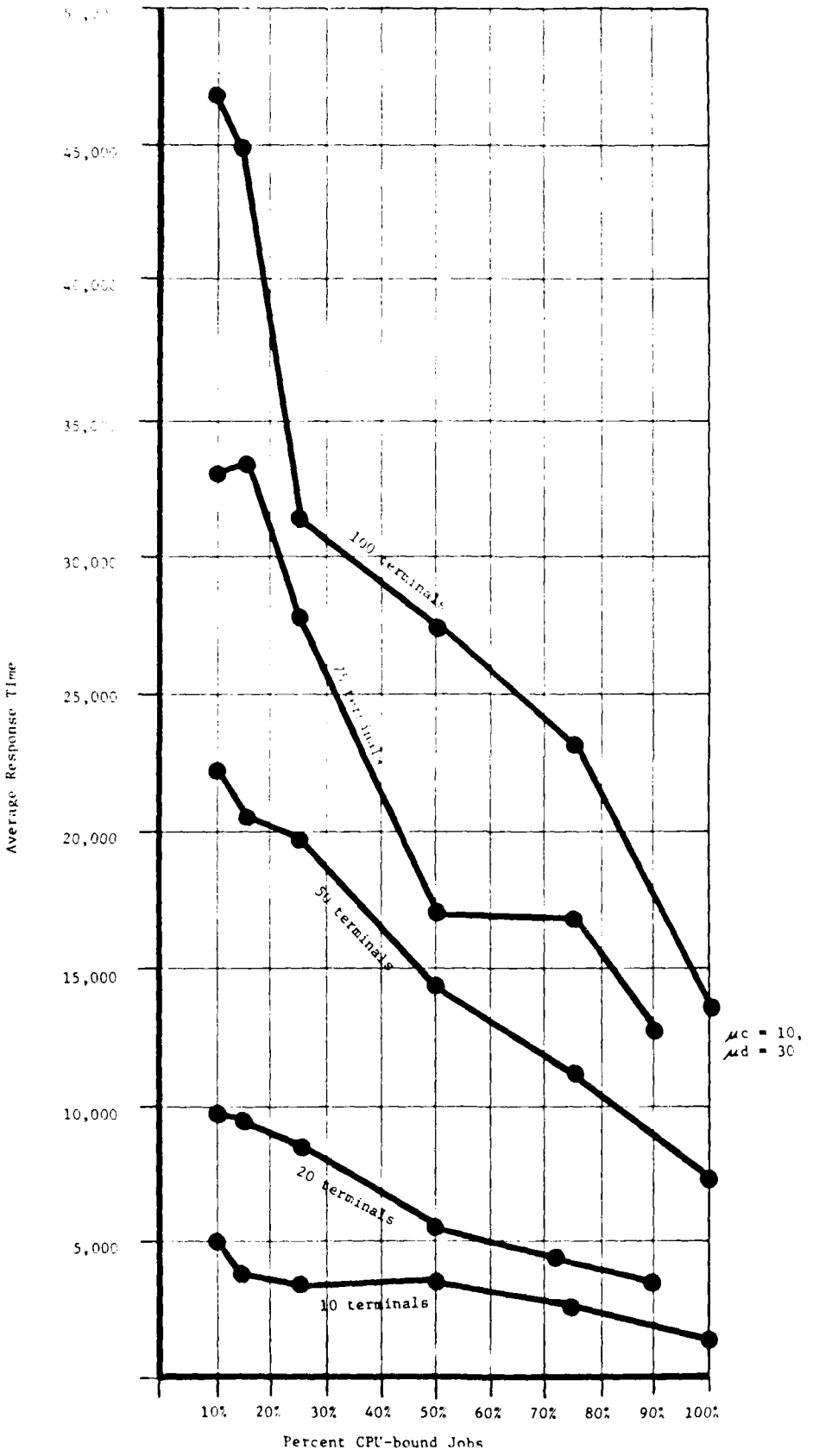


Figure 3-15. Percent CPU-bound Jobs vs. Average Response Time

for a mean CPU/disk iteration count for CPU-bound jobs of 10 and for disk-bound jobs of 30.

Figure 3-16 and 3-17 illustrate the results of increasing the average CPU service time by a factor of 12. Looking only at an operating region represented by 8 to 20 terminals, the average response time is not affected greatly, either for a CPU/disk iteration count of (4,12) or one of (8,24).

The wide range of parameter values for the simulator that would be considered realistic makes it difficult to draw final and definitive conclusions from the experiments that have been conducted. It can be said that having examined a set of cases with a limited set of parameter values, it is clear that response time is proportional to the number of terminals and the number of CPU/disk iterations, and inversely proportional to the percentage of CPU-bound jobs. As far as how these scale factors actually are mathematically interrelated, the curves show that these relationships depend on the operating region, i.e., whether the system is CPU-bound or disk-bound and whether there is a small (maybe 20 or less) or large (more than 50) number of terminals.

Further work to make the simulator more sensitive to the particular requirements of IDHS and to run experiments with additional sets of parameter values would permit more definitive analyses of the scale factor interrelationships. Such results would also enable the scale factor-simulation parameter equations, as described in the earlier report, "Simulator Variable-Scale Factor Equations" (Appendix E), to be completely derived.

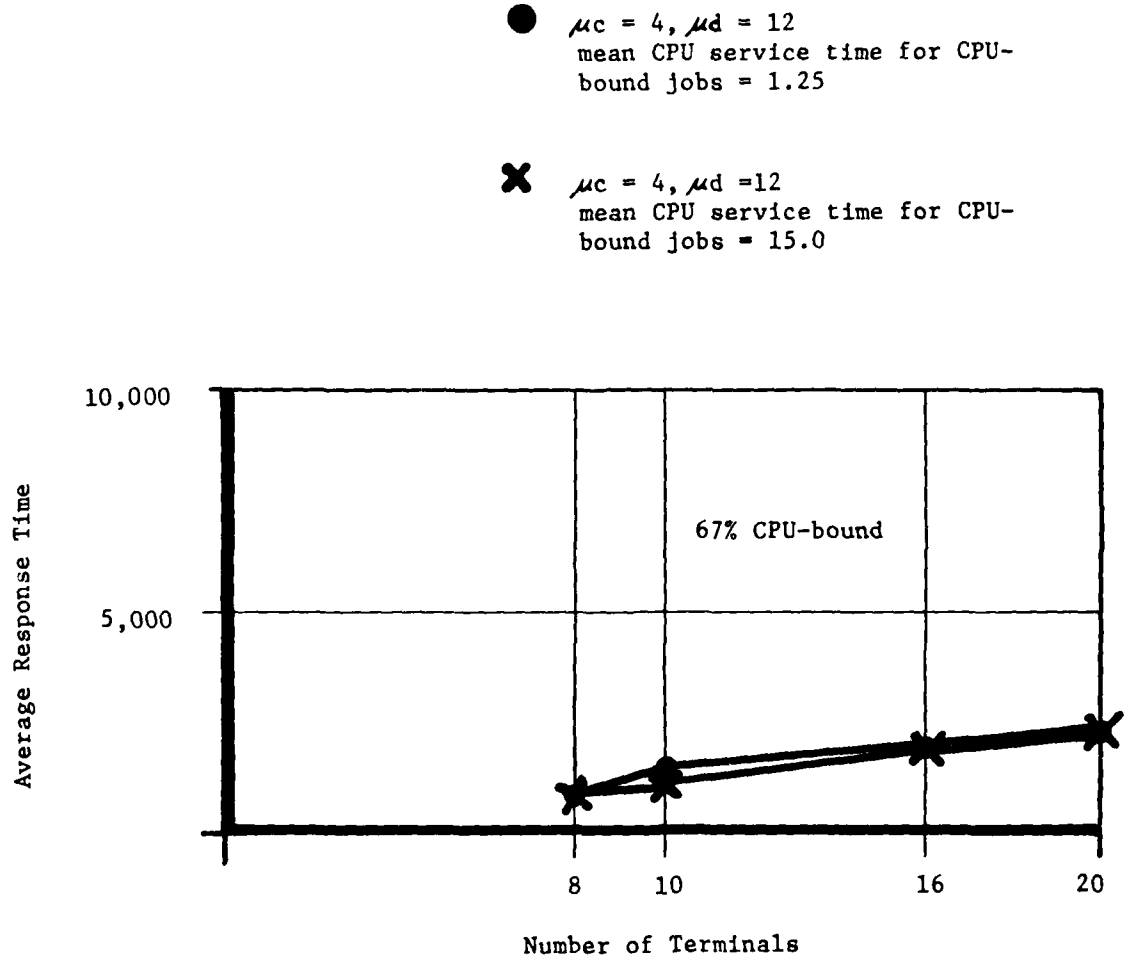


Figure 3-16. Number of Terminals vs. Average Response Time

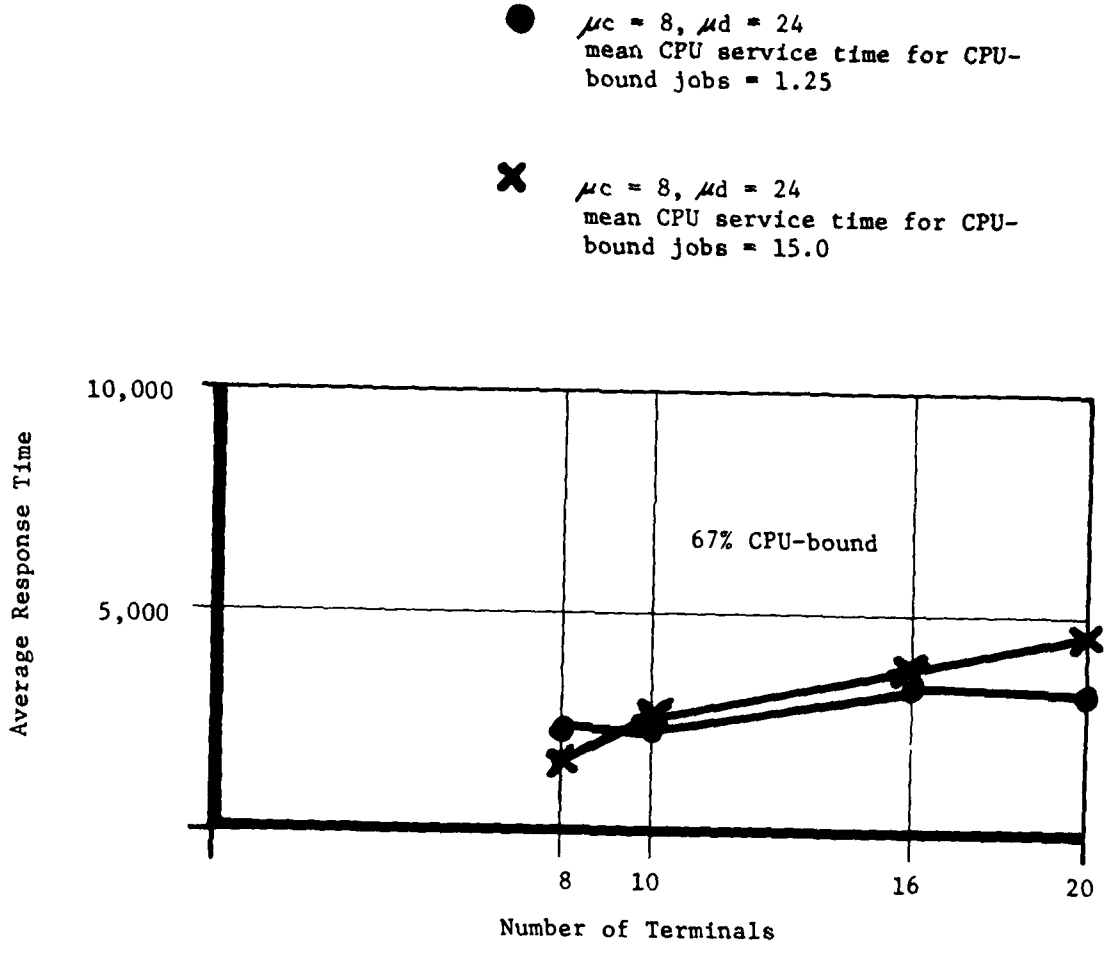


Figure 3-17. Number of Terminals vs. Average Response Time

3.2.2 Guidelines on Which Parameters to Scale

The objective of the research into the scaling of systems before full implementation is attempted is to improve the way design, development, and evaluation of IDHS are performed. The ultimate IDHS is derived from the scaled system in a manner that decreases the final cost and increases the final benefit over that achievable without the use of a scaled system. Consider Figure 3-18, which illustrates the relationship among IDHS, development of IDHS, and scaled systems. Characteristics of intelligence data handling computer systems, when considered in light of what is known today about computer system development, dictate a certain cost/benefit achievable with a given development effort. Suppose that a scaled system is defined, based on the ultimate intelligence data handling system objectives, but without some of the characteristics that contribute to increased cost and reduced benefit. The scaled system could then be implemented at a fraction of the cost of the complete system, and could furthermore be used to change some of the undesirable characteristics of the ultimate system. For example, one factor increasing system cost is lack of personnel experience with the system. After developing a scaled system, project personnel will have the experience necessary to develop the complete system at reduced cost. Thus, the ultimate intelligence data handling computer system is derived from the scaled system in a manner that decreases the final cost and increases the final benefit over that achievable without the use of a scaled system.

The unique problems entailed in implementing an IDHS computer system are based on a combination of its characteristics. Since understanding

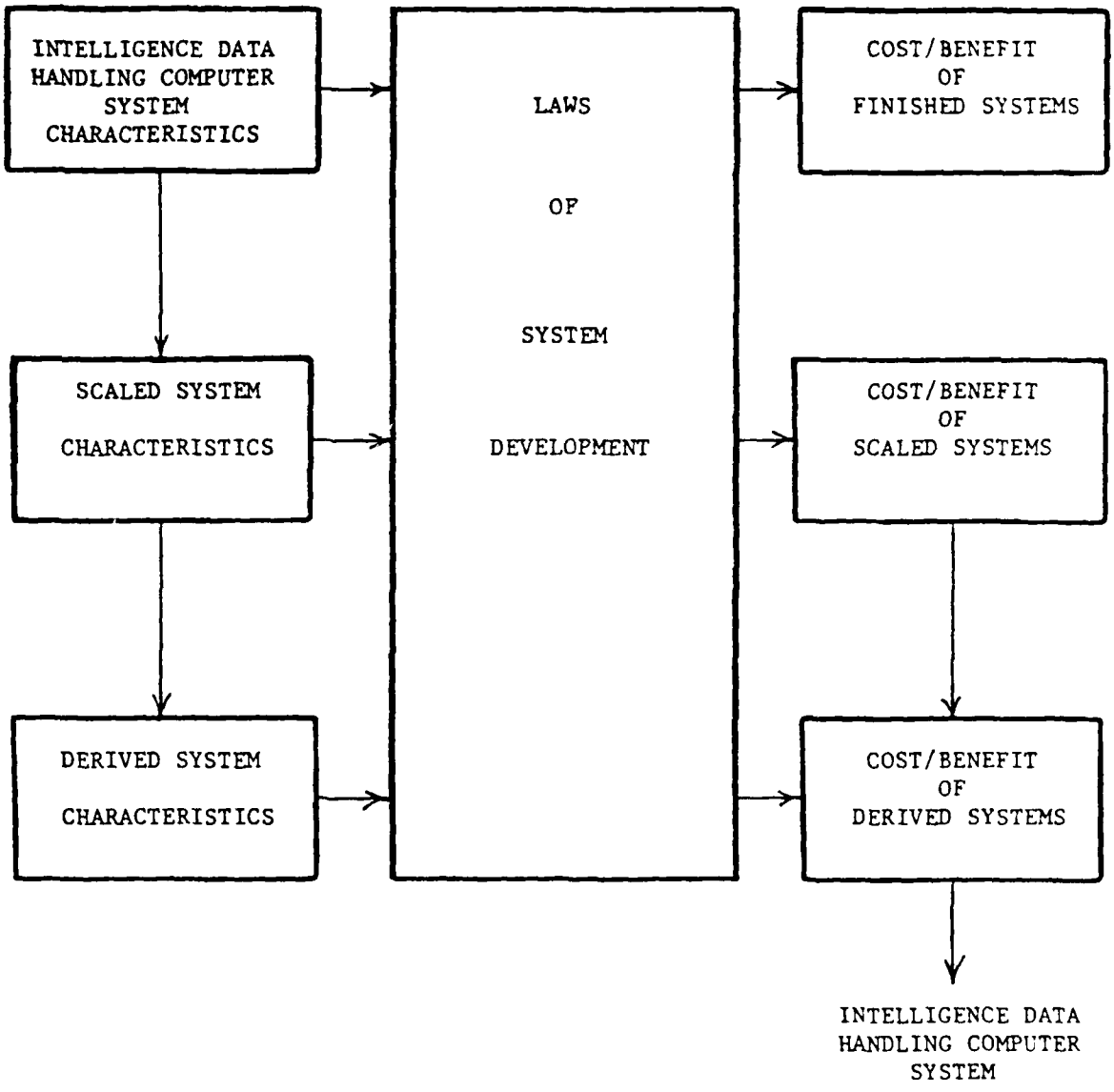


Figure 3-18. Scaled System Utilization for Development

how these characteristics might be modified within a scaled system is necessary to using scaled system techniques, major characteristics of IDHS will be discussed.

a. System uniqueness

Most intelligence data handling computer systems are unique. Although most do share common functions, such as communications, each system developed must support specific mission requirements and interface with specific other in-place systems. Cost savings have been realized through transfer of technology, such as implementing National Military Intelligence Center (NMIC) Support Software (NSS) for the Preliminary Operational Capability (POC) of the Pacific Command (PACOM) Data Services Center (PDSC), but uniqueness is not removed through this process. Thus, several man-years of development were still required for the PDSC POC due to unique hardware interfaces and different computer configurations. In addition, implementation of many new and unique capabilities for PDSC is currently underway.

b. Security

All intelligence data handling computer systems operate within secure environments due to the classified information they process. Many of these systems are subject to the especially stringent security constraints required for processing sensitive compartmented information (SCI). Types of security required include physical (access restriction), personnel (clearances required for access), TEMPEST (electronic emanation), and COMSEC (communications security between systems). In addition, computer hardware/software security provides another line of defense against unauthorized access by preventing information retrieval

without knowledge of correct passwords and codewords even if physical security is breached. Techniques of hardware/software security are expected to improve considerably in the near future, as extremely reliable measures are required to process data of differing classifications within the same system. Current requirements for such multi-level secure processing have spurred research efforts such as Kernelized Secure Operating System (KSOS). Parameters related to security objectives such as file protection methods, granularity of data access control, encryption, and authentication mechanisms are potential elements for scaling.

c. Interactive

Most intelligence data handling computer systems are interactive; that is, they interface with users at on-line terminals. In order to be effective, these systems must provide rapid response to user requests. Many of these requests may require complex processing, and a large number of user terminals is often supported. Thus, the NMIC system may be accessed from over thirty terminals, and may process requests to search an entire five-day message file for specific items. The number of terminals and the required responsiveness can be objects of scaling procedures.

d. Real-Time

In addition to being interactive, most intelligence data handling computer systems also include components that must operate in real-time. This is particularly true for components handling direct sensor input or, as is more common, components handling communication circuitry and protocols. Thus, the Intelligence Data Handling System -

Communications (IDHSC II) is capable of controlling several communication channels with bandwidths of 9600 baud. Messages must be processed as they are received, and must be transmitted with timeliness. IDHSC II additionally performs sophisticated packet switching and other message handling functions, and it is conceivable that bandwidths of up to 50KB may eventually be required. Real-time operations can be scaled by simulating real-time data with input data, and transmission and dissemination functions can be eliminated for scaling purposes.

e. State-of-the-art

Most intelligence data handling computer systems include at least some components that are state-of-the-art. Some systems are based entirely upon research into state-of-the-art techniques. For example, the Advanced Indications System (AIS) includes aspects relating to artificial intelligence and the emerging technology of decision support systems. It would probably not be desirable to scale state-of-the-art features.

f. Large data base

Increasing sophistication in intelligence collection techniques and expanding computer storage and processing capabilities have provided impetus toward development of intelligence data handling computer systems with data bases of ever-increasing size and complexity. The Advanced Imagery Requirements Exploitation System (AIRES) data base currently consists of several billion characters of on-line information. Data base size and the number of intra-data base linkages are prime candidates for scaling.

g. Interoperability

The vast amount of intelligence data collected and the decentralization of mission responsibility, particularly as is being implemented under the Delegated Production Policy, dictate that many different intelligence data handling computer systems exist in diverse geographic locations. However, the necessity for fusion of intelligence from different sources requires that communication and interoperability be established among these various intelligence data handling computer systems. Interoperability requirements are particularly wide-ranging for national-level systems such as the Defense Intelligence Agency (DIA) Integrated Indications Systems (DIIS) currently being designed, which will interface with at least a dozen other systems. Different locations can be scaled by simulating through input data.

h. Reliability/Availability

Many intelligence data handling computer systems operate on an around-the-clock schedule, and all are expected to be available during virtually 100% of their scheduled up-time. With many of these systems extremely critical for the national defense of the United States, serious degradations of reliability and/or availability cannot be tolerated. Also, the extensive amount of interoperability implemented causes systems to depend on each other and may cause one malfunctioning system to adversely impact others.

i. Changing Requirements/Evolving Systems

Intelligence collector technology growth, coupled with the long lead times required to implement data handling computer systems, often causes system requirements to change several times during a development

effort. Furthermore, the overall national intelligence data handling capability is continually evolving, causing each individual intelligence data handling computer system to similarly evolve. The Community On-Line Intelligence System (COINS) provides a good example of national evolution and requirements changes affecting several individual automated systems. COINS was initially implemented as a dedicated network directly interconnecting various large-scale host mainframes. As additional hosts were added to the network, it became apparent that host programming changes were becoming prohibitively expensive, so a front-end processor architecture was implemented. The architecture also included communication processors similar to the Interface Message Processors (IMPs) used on the Advanced Research Projects Agency Network (ARPANET). Some network sites did not have IMPs but did have IDHSC II processors, however, so COINS protocols were implemented through IDHSC II and interfaced to other members of the original COINS. Efforts currently underway with respect to COINS include an experiment to eliminate dedicated circuits by sending traffic, suitably encrypted, to distant sites through the actual ARPANET.

The system parameters which apply to each IDHS characteristic are described more fully in "Software Scale Parameters" (Appendix B) and "System Scale Factor Metrics" (Appendix C). Which of these characteristics to scale depends on the major objectives of the development effort, making it hard to quantify application-dependent parameters. In addition, quantifying software system attributes is a young, expanding discipline in which definitions and emphases tend to shift, contributing to the dynamic nature of the terminology and

technical base. Actual metrics and relationships may therefore be resculptured as research continues toward the goal of achieving an understandable and workable methodology for scaled systems development.

The experiments performed with the simulator indicate that guidelines for which parameters to scale will have to be narrowly defined, depending on such factors as operating region and expected system size, e.g., number of terminals. Although no drastic changes in scale factor interrelationships occur in these different environments, there is a significant amount of variance, e.g., doubling the number of terminals does not always double the response time.

Figure 3-19 illustrates the considerations in deciding what to scale. It is necessary and advantageous to first prepare the lists of objectives for using both the full-scale system and the scaled system. Full-scale system objectives fall into two categories, the general type of system, such as real-time versus batch, and any unique objectives required for the system, such as 100% up-time, flexibility to interface with other evolving systems, simple transportability, etc. For example, a real-time data acquisition system could be scaled on the input data rates or number of input lines, while an interactive system could be scaled on the number of users. The objectives of the full-scale system are related to the functions it performs, which will aid in determining which parameters to scale. The objectives for using scaled systems within a development effort will also be factors in determining which parameters to scale, as scaling certain parameters may clearly aid or hinder accomplishment of these objectives. As has been discussed in previous sections, the benefits include obtaining user feedback for final

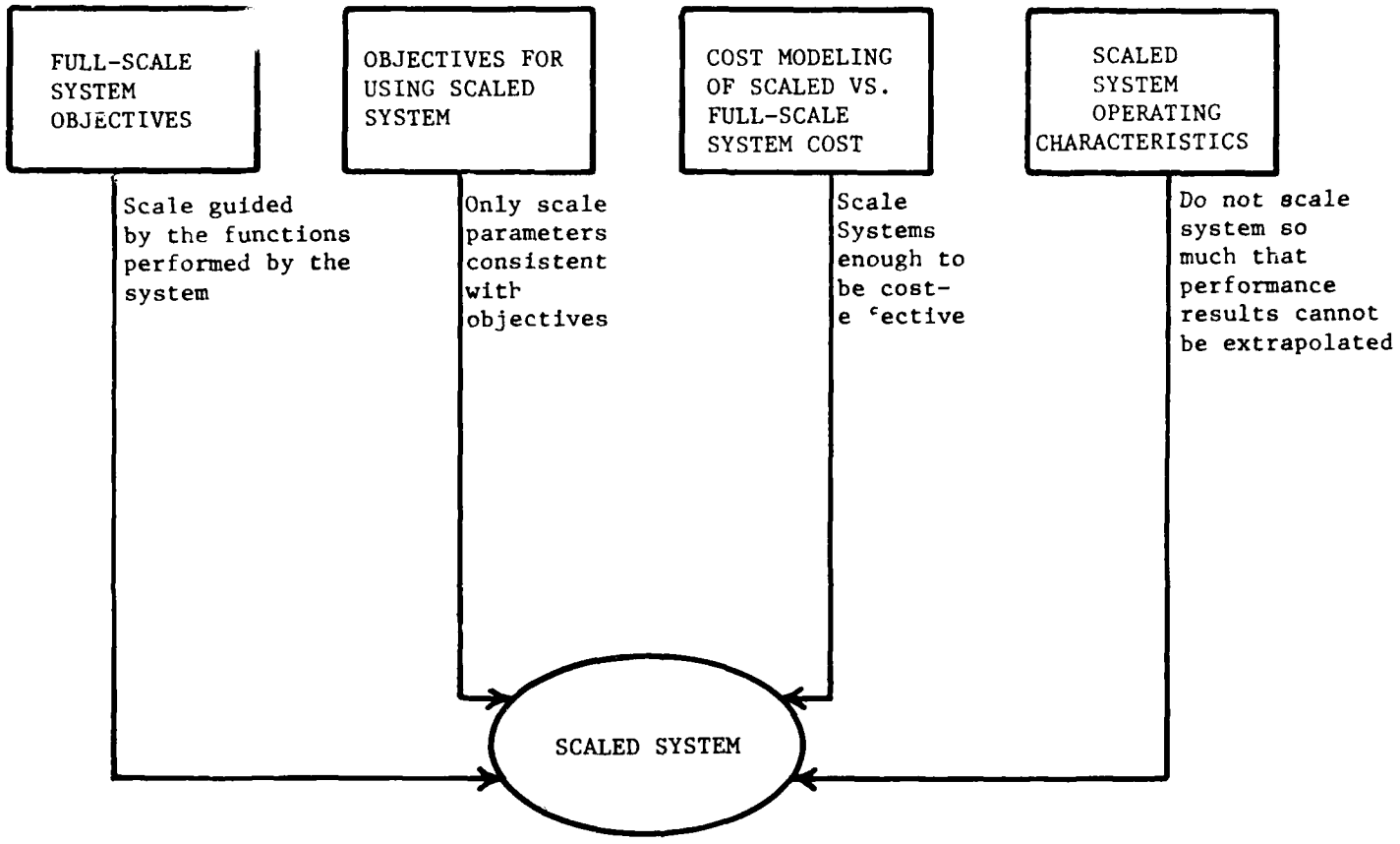


Figure 3-19. Scaling Considerations

design decisions, testing unique or state-of-the-art concepts, providing project experience for the development team, implementing an initial operational capability which will be later enhanced, and predicting full-scale system cost, schedule, risk, and performance. A scaled system built to establish the feasibility for a unique system should not have complex or state-of-the-art features scaled, while a scaled system built to elicit final user design feedback should have the number of users, but not the user interface, scaled. The use of a scaled system must also be cost-effective, while "too much" scaling must be avoided or it will be impossible to extrapolate performance results. Figure 3-20 presents a summary of guidelines in selecting which sample objectives to scale.

The value of using the simulator to aid in determining guidelines as to which parameters to scale is that, for a given set of objectives for the full-scale system, tests can be run with various scenarios representing different sets of parameters scaled and the implications of such scaling can be easily and inexpensively evaluated. The limits beyond which some scale factors should not be scaled can also be determined in this way. For example, it might be seen that given the set of parameter values that define the proposed system, halving the number of terminals from 100 to 50 doubles the interactive responsiveness. However, halving the number of terminals from 50 to 25 triples the interactive responsiveness. In this case, the simulator would indicate that the number of terminals should not be scaled to less than half without taking the change in the terminal responsiveness relationship into account.

SAMPLE OBJECTIVES	PARAMETERS TO SCALE
System Uniqueness	?
Security	File protection methods, granularity of data access control, encryption, authentication
Interactive	Number of terminals, required responsiveness
Real-Time	Input data rates, number of input lines, transmission and dissemination functions
State-Of-The-Art	?
Large Data Base	Data base size, number of intra-data base linkages
Interoperability	Transmission and dissemination functions
Reliability/Availability	?

Figure 3-20. Summary of Scaling Guidelines

General guidelines can be derived, however, from this research. The experiments have demonstrated that responsiveness is inversely proportional to the number of terminals and the number of CPU/disk iterations. It can also be seen that, as shown in Figures 3-15 and 3-16, increasing the average CPU time does not increase the response time, indicating that functions requiring extra CPU time, e.g., security overhead, can be moderately scaled without affecting other scale factors. The cost model can then be used to determine the degree of scaling that is both advantageous and feasible.

3.3 Decision Factors and Guidelines

The purpose of this section is to establish the basic and generalized guidelines which system architects can refer to in determining the feasibility and cost-effectiveness of building a proposed system to scale. From that point, a more detailed discussion supported by quantitative exhibits will be presented.

3.3.1 Overview

Decision guidelines for potential scaling of proposed system designs will most often have as their focus, two major questions:

(1) Can the system in question be built to scale?

(2) Will the resultant scaled system prove worthwhile both in its possible operational value and in benefits realized for application to the unscaling effort?

3.3.1.1 Scaling Feasibility

Before answering the first question, a thorough analysis of the proposed system's characteristics must be performed in order to establish a reasonable scaling methodology. Information of value would generally

consist of such items as the system's requirements, performance criteria, functionality, proposed architecture, estimated program and data base size, and configuration, both in terms of its hardware and software. In addition, level of technical staff experience and qualifications, development schedule, resource allocation (staff-loading), as well as the ultimate target delivery date for the full-scale system must be taken into account. Once such data is gathered, the formulation of a scaled system development methodology may commence. Coupled with the "how" of scaling, however, is the "why" of scaling, which raises the importance of the second question stated.

It should be stated that the importance of a scaled system lies not in the fact that the scaling can actually be accomplished, but in the benefits that actually accrue to the ultimate full-scale system. It is important then, that the objectives of the scaled system be established early on. It is additionally important to maintain the distinction between the concepts of scaling and prototyping. While a scaled system is most certainly a prototype, a prototype may not necessarily have the properties of a scaled system. While the potential value of prototype systems is acknowledged, the discussion of such is considered beyond the scope of this report.

In examining system attributes in terms of scaling feasibility, the ones with the least risk should be considered first. The motivation here is to scale attributes where there is much certainty about their full-scale properties so that relatively higher-risk system components may be implemented and thoroughly scrutinized in the scaled system.

Important attributes to keep in mind during the design of a scaled system include its degree of modularity and transportability. In most cases, the unscaling effort would certainly benefit from any inventory of source code accumulated during the scaled effort. This, of course, requires the extra effort in planning since little is known about the actual workings of the full-scale system at the front-end of the development cycle and, due to the development methodology selected (scaling), it is most certainly a complex and technically challenging system defying any such planning attempts. Nevertheless, attention to designing modular, transportable code for the scaled system will eliminate the need to produce similar code for the ultimate full-scale system and will result in cost savings for the full-scale system as well as a reduction in total project costs.

3.3.1.2 Cost Modeling and Parametric Analysis

In the planning and design stages for a scaled system, the planners inevitably find themselves deep in the realm of cost estimation modeling and parametric analysis in the determination of the potential cost effectiveness of the development methodology chosen. Such tools are important in the exploration of the interrelationships that exist between the scaled system and its full-scale counterpart in determining total cost, schedule, and risk. While hardware cost estimation can be achieved with an acceptable degree of accuracy, software cost estimation involves many critical variables which aggravate formulation of accurate cost projections. One such variable is time; major software development efforts nearly always span a considerable amount of time. Software cost estimates therefore bear a significant degree of uncertainty because they

address future events heavily dependent upon the interaction of a group of people. Consequently, a small deviation in the resulting delivery schedule causes a major impact upon costs because in software development, the burdened costs of maintaining a project staff, generally at significant pay-scales, are large. Future projections thus bear a degree of uncertainty proportional to the term under consideration; long-term predictions are long on risk while shorter-term predictions involve relatively less risk. System hardware cost estimation is considered a contrast to software cost estimation because, in the procurement of hardware, the objects are generally "off-the-shelf" items where the major concerns deal mostly with the transportation, interfacing, and check-out of the various modular hardware components and a relatively shorter time frame is involved.

Due to the difficulty involved in dealing with critical variables, such as time, in the planning of systems, parametric analysis has become a useful tool in the making of projections. Parametric analysis can be loosely defined, for the purposes of this discussion, as the posing of "what if" questions; the power of the technique lies in its assessment of the sensitivities of the various crucial variables present in our estimate calculations, such as time.

3.3.1.3 Scaled Systems Decision Criteria

Thus, while the feasibility and methodology of producing a system to scale is the primary responsibility of the system's architects, the resources of a parametric analyst and a cost estimation method are crucial in determining, at the onset, any potential cost savings that could occur through the adoption of a scaled system development

methodology. Cost savings are perceived as the principal driver of the scaled system design; however, it should be emphasized that situations may arise where potential cost savings are subordinate to full-scale product quality considerations such as reliability, efficiency, integrity, and performance. Nevertheless, the objective of the following sections is to provide the system planner with appropriate guidelines by which he may mentally determine the feasibility of applying the scaled systems approach to a particular software effort.

3.3.2 Decision Factors Influencing System Development

There are a number of research papers appearing in the open literature itemizing factors which influence software development cost and schedule. Some authors have additionally been able to quantify the effects of the presence or absence, to varying degrees, of these factors. One of the first to do so was J.D. Aron in "Estimating Resources for Large Programming Systems" [ref. 1]. A result of this study is illustrated in Figure 3-21. In this illustration, we find Aron's productivity table which relates code production to factors such as difficulty, schedule duration, and interface complexity. Of note to planners of scaled systems are the facts that, generally, the longer the development schedule duration and the less interface complexity and difficulty, the greater the productivity and, hence, the less the cost. Of especial interest is the counter-intuitive nature of productivity presented in terms of development schedule duration; the longer the schedule, the greater the productivity. This anomaly has been noted by other authors such as Brooks and Putnam and the phenomenon is perhaps best explained by Putnam [ref. 17]. Yet even from this simple table,

	Duration / Difficulty	6-12 Months	12-24 Months	More Than 24 Months	
Row 1	Easy	20	500 (25/day)	10,000 (40/day)	Very Few Interactions
Row 2	Medium	10	250 (12.5/day)	5,000 (20/day)	Some Interactions
Row 3	Difficult	5	125 (6.25/day)	1,500 (6/day)	Many Interactions
	Units	Instructions per Man-Day	Instructions per Man-Month	Instructions per Man-Year	

Figure 3-21. Aron's Productivity Table

planners of scaled systems can be confident that through the limiting of a project's size, scope, and complexity - some of the attributes of a scaled system - productivity performance can indeed be increased and total resources and labor put to more efficient use.

Next to itemize software developmental factors was Doty (and associates) under a research effort for the Rome Air Development Center (RADC). In Software Cost Estimating Study - Guidelines for Improved Software Cost Estimating [ref. 7], the authors identified forty-six factors which contribute significant impacts upon software project costs and schedules. These forty-six factors were divided into three homogeneous groups and are listed in Figure 3-22. In addition to this enumeration, Doty and his associates were able to formulate a set of effort (cost) formulae characterized by separations based upon application type and respective adjustment factors specifically accounting for some of the environmental attributes. These formulae were arrived at based upon the data RADC had internalized concerning over four hundred software development efforts. The Doty cost formulae and adjustment factors appear in Figure 3-23. Individual environmental factors quantified in Figure 3-23 are identified in Figure 3-22 by an asterisk ("*") alongside the corresponding factor. It is readily apparent that not all of the effects of the factors listed in Figure 3-22 were quantified. Presumably this is due to the inherent difficulty and probable research constraints limiting the quantitative determination of such effects.

Of wide interest to researchers of software engineering in general, and cost and productivity modeling in particular, is an article entitled

REQUIREMENTS DOMAIN FACTORS

*	1.	OPERATIONAL REQUIREMENTS DEFINITION
*	2.	OPERATIONAL REQUIREMENTS CHANGES
	3.	USER REQUIREMENTS CONSIDERED
	4.	OPERATIONAL REQUIREMENTS/DESIGN INTERFACE
	5.	SPECIFIED RESPONSE TIME
	6.	AVIONICS APPLICATION
*	7.	COMMAND AND CONTROL APPLICATION
*	8.	MULTIPLE SOFTWARE UTILIZATION SITES
	9.	RELIABILITY REQUIREMENTS
	10.	MAINTAINABILITY REQUIREMENTS
	11.	QUALITY REQUIREMENTS
	12.	TRANSPORTABILITY REQUIREMENTS
*	13.	BUSINESS APPLICATION
*	14.	SCIENTIFIC APPLICATION
*	15.	UTILITY APPLICATION

SYSTEM ARCHITECTURE/ENGINEERING (A/E) FACTORS

*	1.	CPU TIME CONSTRAINED
*	2.	PROGRAM MEMORY SIZE CONSTRAINED
*	3.	ON-LINE OPERATION
	4.	TIME AND MEMORY CONSTRAINED
	5.	TARGET CPU DESIGNATION
	6.	DESIGN STABILITY
	7.	DESIGN COMPLEXITY

* Productivity impacts quantified in study

Figure 3-22. RADC Environmental Factors

MANAGEMENT DOMAIN FACTORS

1. SUPPORT SOFTWARE AVAILABILITY
2. WORK BREAKDOWN STRUCTURE
3. DEGREE OF INNOVATION
4. TESTING REQUIREMENTS INCLUDING VERIFICATION AND VALIDATION
5. COST/SCHEDULE CONTROL SYSTEMS CRITERIA (C/SCSC)
6. DEVELOPMENT PERSONNEL MIX
7. PROGRAMMER TESTING
8. AMOUNT & METHOD OF COST DATA COLLECTION
9. COST OF SECONDARY RESOURCES
10. DEFINITION OF INSTRUCTION
11. SIZING ERROR
12. DATA MANAGEMENT TECHNIQUES
13. MODERN PROGRAMMING TECHNIQUES
- ** 14. PROGRAMMING FACILITIES (Location & Access)
- * 15. DIFFERENT DEVELOPMENT AND TARGET COMPUTERS
16. COMMUNICATIONS
17. LANGUAGE REQUIREMENTS
- * 18. DEVELOPMENT SITE
- * 19. DEVELOPER USING ANOTHER ACTIVITY'S COMPUTER
- * 20. NUMBER OF DEVELOPMENT LOCATIONS
- * 21. CONCURRENT DEVELOPMENT OF HARDWARE
- * 22. DEVELOPER'S FIRST TIME ON SPECIFIED COMPUTER
- * 23. SPECIAL DISPLAY REQUIREMENTS
24. SOFTWARE DEVELOPMENT SCHEDULE

* Productivity impacts quantified in study

Figure 3-22 (Cont.)

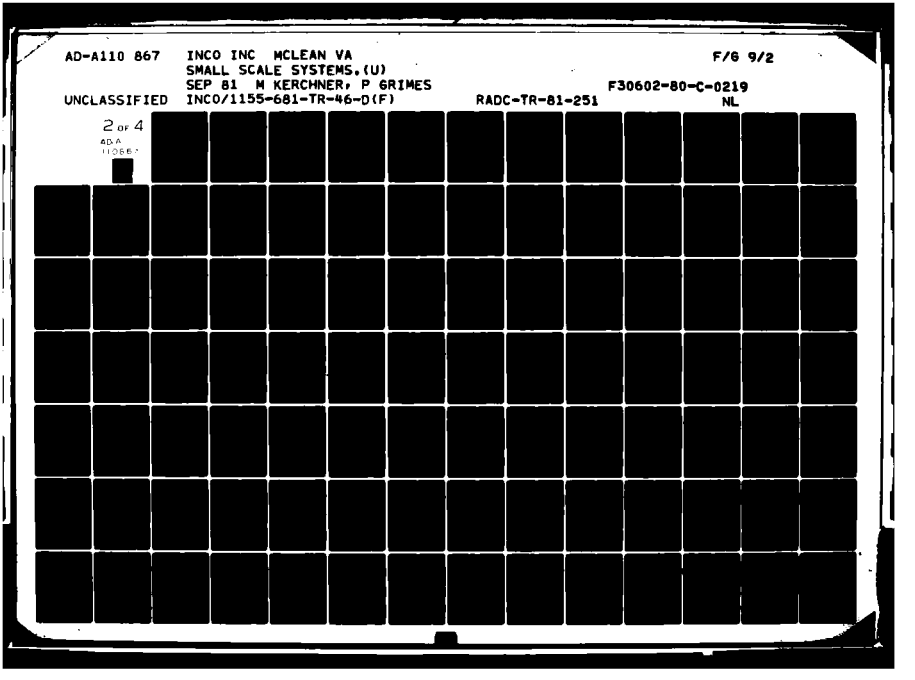
ESTIMATOR FOR PM: / FACTOR		APPLICATIONS									
		ALL		COMMAND & CONTROL		SCIENTIFIC		BUSINESS		UTILITY	
		$2.060 \left[1.047^{\sum_{j=1}^{j=14} R_j f_j}\right]$		$0.591 \left[1.263^{\sum_{j=1}^{j=14} R_j f_j}\right]$		$2.011 \left[1.019^{\sum_{j=1}^{j=14} R_j f_j}\right]$		$3.742 \left[2.781^{\sum_{j=1}^{j=14} R_j f_j}\right]$		$1.744 \left[0.811^{\sum_{j=1}^{j=14} R_j f_j}\right]$	
		YES	NO	YES	NO	YES	NO	YES	NO	YES	NO
SPECIAL DISPLAY	f_1	1.11	1.00	1.11	1.00	1.11	1.00	1.43	1.00	1.00	1.00
DETAILED DEFINITION OF OPERATIONAL REQUIREMENTS	f_2	1.00	1.11	1.00	1.54	1.00	2.00	1.00	1.00	1.00	1.00
CHANGE TO OPERATIONAL REQUIREMENTS	f_3	1.05	1.00	1.05	1.00	1.05	1.00	1.05	1.00	1.05	1.00
REAL TIME OPERATION	f_4	1.33	1.00	1.33	1.00	1.67	1.00	1.00	1.00	1.43	1.00
CPU MEMORY CONSTRAINT	f_5	1.43	1.00	1.25	1.00	1.25	1.00	1.00	1.00	1.18	1.00
CPU TIME CONSTRAINT	f_6	1.33	1.00	1.51	1.00	1.67	1.00	1.00	1.00	2.32	1.00
FIRST S/W DEVELOPED ON CPU	f_7	1.92	1.00	1.92	1.00	1.92	1.00	1.92	1.00	1.92	1.00
CONCURRENT DEVELOPMENT OF ADP H/W	f_8	1.82	1.00	1.47	1.00	2.22	1.00	1.33	1.00	1.25	1.00
TIME SHARE, VIS-A-VIS BATCH PROCESSING IN DEVELOPMENT	f_9	0.83	1.00	0.83	1.00	0.83	1.00	0.83	1.00	0.83	1.00
DEVELOPER USING COMPUTER AT ANOTHER FACILITY	f_{10}	1.43	1.00	1.43	1.00	1.43	1.00	1.43	1.00	1.43	1.00
DEVELOPMENT AT OPERATIONAL SITE	f_{11}	1.39	1.00	1.39	1.00	1.39	1.00	1.39	1.00	1.39	1.00
DEVELOPMENT COMPUTER DIFFERENT THAN TARGET COMPUTER	f_{12}	1.25	1.00	2.22	1.00	1.11	1.00	1.00	1.00	1.43	1.00
DEVELOPMENT AT MORE THAN ONE SITE	f_{13}	1.25	1.00	1.75	1.00	1.75	1.00	1.25	1.00	1.21	1.00
PROGRAMMER ACCESS TO COMPUTER	f_{14}	LIM: 1.00 UNLIM: 0.90		1.00 1.00		1.00 0.67		1.00 0.90		1.00 0.67	

Figure 3-23. RADC/Doty Factors Affecting Development

"A Method of Programming Measurement and Estimation," by Felix and Walston of IBM [ref. 24]. This article first appeared in the IBM System Journal Volume 16, Number 1, in 1977. In this article, the authors examined a group of sixty completed software development projects that covered a wide range of application type, size, and complexity. From this research, the authors compiled a list of productivity rates itemized by environmental or product factor. This list is summarized in Figure 3-24. Regrettably, the data, as presented, is not of much use to system planners. The authors did allude to a methodology whereby the productivity rates could be incorporated into an estimation model but unfortunately they did not elaborate upon the details necessary to apply the methodology to practice. Hence, under this research effort the attempt was made to incorporate this raw data into a general scheme of guidelines through which system planners might be able to assess the potential benefits of applying the scaled systems development methodology.

3.3.2.1 Factor Quantification

The Walston and Felix article is one of the few available sources of quantitative empirical data concerning the effects of many various environmental factors influencing software development. In order to obtain meaningful decision factors from the Walston and Felix data identified in the previous section, it was first necessary to somehow translate the raw productivity rates into some sort of predictive coefficients indicating the respective impacts of the development factors on a project's cost, effort, or schedule. While it was recognized that the resultant factors may not apply to any particular environment, the


AD-A110 867
INCO INC MCLEAN VA
SMALL SCALE SYSTEMS,(U)
SEP 81 M KERCHNER, P GRIMES
F/G 9/2
F30602-80-C-0219
UNCLASSIFIED
INCO/1155-681-TR-46-D(F)
RADC-TR-81-251
NL
2 OF 4
ADA
110867

FACTOR	RESPONSE GROUP MEAN PRODUCTIVITY (DSL/PM)		
STRUCTURED PROGRAMMING	0 - 33% 169	34 - 66% -	>66% 301
DESIGN AND CODE INSPECTIONS	0 - 33% 220	34 - 66% 300	>66% 339
TOP DOWN DEVELOPMENT	0 - 33% 196	34 - 66% 237	>66% 321
CHIEF PROGRAMMER TEAM USAGE	0 - 33% 219	34 - 66% -	>66% 408
OVERALL COMPLEXITY OF CODE DEVELOPED	<AVERAGE 314		>AVERAGE 185
COMPLEXITY OF APPLICATION PROCESSING	<AVERAGE 349	AVERAGE 345	>AVERAGE 168
COMPLEXITY OF PROGRAM FLOW	<AVERAGE 289	AVERAGE 299	>AVERAGE 209
OVERALL CONSTRAINTS IN PROGRAM DESIGN	MINIMAL 293	AVERAGE 286	SEVERE 166
PROGRAM DESIGN CONSTRAINTS ON MAIN STORAGE	MINIMAL 391	AVERAGE 277	SEVERE 193
PROGRAM DESIGN CONSTRAINTS ON TIMING	MINIMAL 303	AVERAGE 317	SEVERE 171
CODE FOR REAL-TIME OR INTERACTIVE OPERATION, OR EXECUTING UNDER SEVERE TIMING CONSTRAINT	<10% 279	10 - 40% 337	>40% 203
PERCENTAGE OF CODE FOR DELIVERY	0 - 90% 159	91 - 99% 327	100% 265
CODE CLASSIFIED AS NONMATHEMATICAL APPLICATION AND I/O FORMATTING PROGRAMS	0 - 33% 188	34 - 66% 311	67-100% 267
NO. OF CLASSES OF ITEMS IN THE DATA BASE PER 1000 LINES OF CODE	0 - 15 334	16 - 80 243	>80 193
NO. OF PAGES OF DELIVERED DOCUMENTATION PER 1000 LINES OF DELIVERED CODE	0 - 32 320	33 - 88 252	>88 195

FACTOR	RESPONSE GROUP MEAN PRODUCTIVITY (DSL/PM)		
CUSTOMER INTERFACE COMPLEXITY	<NORMAL 500	NORMAL 295	>NORMAL 124
USER PARTICIPATION IN THE DEFINITION OF REQUIREMENTS	NONE 491	SOME 267	MUCH 205
CUSTOMER ORIGINATED PROGRAM DESIGN CHANGES	FEW 297		MANY 196
CUSTOMER EXPERIENCE WITH THE APPLICATION AREA OF THE PROJECT	NONE 318	SOME 340	MUCH 206
OVERALL PERSONNEL EXPERIENCE AND QUALIFICATIONS	LOW 132	AVERAGE 257	HIGH 410
PERCENTAGE OF PROGRAMMERS DOING DEVELOPMENT WHO PARTICIPATED IN DESIGN OF FUNCTIONAL SPECIFICATIONS	<25% 153	25 - 50% 242	>50% 391
PREVIOUS EXPERIENCE WITH OPERATIONAL COMPUTER	MINIMAL 146	AVERAGE 270	EXTENSIVE 312
PREVIOUS EXPERIENCE WITH PROGRAMMING LANGUAGES	MINIMAL 122	AVERAGE 225	EXTENSIVE 385
PREVIOUS EXPERIENCE WITH APPLICATION OF SIMILAR OR GREATER SIZE AND COMPLEXITY	MINIMAL 146	AVERAGE 221	EXTENSIVE 410
RATIO OF AVERAGE STAFF SIZE TO DURATION (PEOPLE/MONTH)	<0.5 305	0.5-0.9 310	>0.9 173
HARDWARE UNDER CONCURRENT DEVELOPMENT	NO 297		YES 177
DEVELOPMENT COMPUTER ACCESS, OPEN UNDER SPECIAL REQUEST	0% 226	1 - 25% 274	>25% 357
DEVELOPMENT COMPUTER ACCESS, CLOSED	0 - 10% 303	11 - 85% 251	>85% 170
CLASSIFIED SECURITY ENVIRONMENT FOR COMPUTER AND 25% OF PROGRAMS AND DATA	NO 289		YES 156

Figure 3-24. IBM Factors Affecting Development

intent was rather to formulate a set of surrogate values based upon actual "real-world" experience for the purpose of rough effort approximating and scaled system trade-off analysis. Again, the purpose of such an exercise would be to provide the system planner with a tool to facilitate his assessment of the feasibility of adopting the scaled system developmental approach.

After rationalizing that the Doty coefficients must have been based upon similar productivity data as that which Walston and Felix provide (except obtained from a different source - RADC), it was determined that the Doty method would be an attractive model to base the determination of the coefficients from the Walston and Felix data upon. In addition, there would be benefits to representing both sets of data in the same manner as they would compliment each other. In retrospect, the Doty method to account for environmental factors consisted of coefficients that, when multiplied together, produced a multiplicative factor that could be used in an equation of the form:

Person months of Effort = Constant * SLOC ↑ Exponent * ii

where: "ii" is the multiplicative factor

Of particular value in the Doty method is the fact that while each environmental factor value not only relates its marginal impact upon a project's estimated cost, it is expressed in a form such that its implied interrelationship with the other factors is automatically accounted for. Other organizations and researchers have used these same environmental factors and their corresponding coefficients for other, different estimating purposes in their original form with acceptable degrees of success. A prime example would be the Space and Missile Systems

Organization's (SAMSO) Software Programs Office (SPO) in Los Angeles, California, where the Doty factors and Coefficients are used to adjust the technology constant in Putnam's software equation [ref. 13]. The Putnam equation relates system size to total project effort and schedule through the technology constant and is totally different from the Doty methodology. The problem, then, was to quantify the IBM data in a similar manner to the Doty methodology. Traditional systems thinking techniques were applied to the problem first - problem solution through problem decomposition. Step one consisted of combining related development factors into groups. The resulting groupings are shown in Figure 3-25. Of concern to this research was the fact that the aggregation of the effects of the individual development factors tended to over-emphasize the resulting productivity estimates. It was subsequently determined that the original data did not result from "pure" laboratory research conditions and that the mere presence of some environmental factors implied other, related factors. For example, the IBM data might be interpreted to suggest that the presence of both structured programming techniques (303 Lines of Code per Month - LOC/M) and top-down design (319 LOC/M) would result in productivity of 622 source lines per month, which, from the other data present, seems questionable. Top-down development and structured programming techniques, in all probability, occurred simultaneously in the Walston and Felix project data base; hence, additive-type analytical techniques of the published data would tend to over-emphasize the effects of the various development factors. This simplistic example illustrates the problem of attempting to aggregate the resultant effects, in terms of

"Stuctured Techniques"
- Structured Programming
- Design and Code Inspections
- Top-down Development
- Chief Programmer Teams

"Complexity"
- Overall Code Complexity
- Complexity of Application
- Complexity of Program Control Flow

"Code Mix"
- Proportion of Code classed as Non-Mathematical and I/O Formatting
- Proportion R/T, Interactive, or Time-Critical Code
- Proportion of Code Intended for Delivery

"Utilization"
- Overall Program Design Constraints
- Core Memory Design Constraints
- Execution Time Design Constraints

"Platform"
- Customer Interface Complexity
- Degree of User Participation in Req'mts. Def.
- Degree of Customer-Originated Design Changes
- Degree of Customer Experience in Application Area

"Resources"
- Ave. Personnel Experience and Qualifications
- % Dev'mt. Programmers who Participated in Func. Design Spec.
- % Utilization of Currently-available Hardware
- Degree of Previous Experience with Oper. Computer
- Degree of Previous Experience with Programming Languages
- Degree of Previous Experience with Appl. Size and Complexity

"Security"
- Spcl. Req. for Access to Dev'mt. CPU
- Amount of Open Access Time to Dev'mt. CPU
- Classified Security Environment for CPU and 25% of Programs & Data

"Misc. Items"
- Proportion of Data Base Class-Items to 1,000 LOC
- Proportion of Doc. Pages to 1,000 LOC
- Ratio of Staff Size to Project Duration (People/Month)

Figure 3-25. IBM/Walston and Felix Environmental and Product Factor Groupings

productivity, of the various factors in determining guidelines based upon such data. The real problem with the data, as we perceive it, is that it does not result from purely controlled situations. Of course, it is not expected to as it is recognized that the gathering of such data under pure laboratory conditions is far too expensive and time consuming, even if it were possible. The task to be performed then was perceived as inferring, through some quantitative basis, the effects of the combined environmental factors. This was first applied through quantifying the aggregated effects of all the factors in each particular group of related factors. To accomplish this, the extreme low- and high-end productivity rates for each component of a group were totaled. A marginal group productivity impact was then calculated based upon these totals through the following equation:

$$[3.2\text{-a}] \qquad \frac{\text{High total} - \text{low total}}{\text{low total}} = \begin{matrix}\text{Marginal}\\ \text{Aggregate}\\ \text{Productivity}\\ \text{Impact for Group}\end{matrix}$$

The marginal productivity impacts of each group, and the data used in arriving at them, are illustrated in Figure 3-26. From this illustration we see that the components of the group "Structured Programming" contribute positively to productivity by a factor of 1.7. The fact that this translates to a 70% increase in productivity for all organizations and environments is undeterminable; however, in the IBM development arena for a similar set of projects as those which constitute the IBM data, it would be reasonable to expect that these practices would contribute to a

	Group Productivity Impact
Stuctured Techniques	1.70
Structured Programming	
Design and Code Inspections	
Top-down Development	
Chief Programmer Teams	
Complexity	1.69
Overall Code Complexity	
Complexity of Application	
Complexity of Program Control Flow	
Code Mix	1.47
Proportion of Code classed as Non-Mathematical and I/O Formatting	
Proportion R/T, Interactive, or Time-Critical Code	
Proportion of Code Intended for Delivery	
Utilization	1.86
Overall Program Design Constraints	
Core Memory Design Constraints	
Execution Time Design Constraints	
Platform	1.92
Customer Interface Complexity	
Degree of User Participation in Req'mts. Def.	
Degree of Customer-Originated Design Changes	
Degree of Customer Experience in Application Area	
Resources	2.73
Ave. Personnel Experience and Qualifications	
% Dev'mt. Programmers who Participated in Func. Design Spec.	
% Utilization of Currently-available Hardware	
Degree of Previous Experience with Oper. Computer	
Degree of Previous Experience with Programming Languages	
Degree of Previous Experience with Appl. Size and Complexity	
Security	1.72
Spcl. Req. for Access to Dev'mt. CPU	
Amount of Open Access Time to Dev'mt. CPU	
Classified Security Environment for CPU and 25% of Programs & Data	
Misc. Items	* Not Calculated *
Proportion of Data Base Class-Items to 1,000 LOC	
Proportion of Doc. Pages to 1,000 LOC	
Ratio of Staff Size to Project Duration (People/Month)	

Figure 3-26. IBM/Walston and Felix Marginal Group Productivity Impacts

	Group Productivity Impact
Resources	2.73
Ave. Personnel Experience and Qualifications	
% Dev'mt. Programmers who Participated in Func. Design Spec.	
% Utilization of Currently-available Hardware	
Degree of Previous Experience with Oper. Computer	
Degree of Previous Experience with Programming Languages	
Degree of Previous Experience with Appl. Size and Complexity	
Platform	1.92
Customer Interface Complexity	
Degree of User Participation in Req'mts. Def.	
Degree of Customer-Originated Design Changes	
Degree of Customer Experience in Application Area	
Utilization	1.86
Overall Program Design Constraints	
Core Memory Design Constraints	
Execution Time Design Constraints	
Security	1.72
Spcl. Req. for Access to Dev'mt. CPU	
Amount of Open Access Time to Dev'mt. CPU	
Classified Security Environment for CPU and 25% of Programs & Data	
Stuctured Techniques	1.70
Structured Programming	
Design and Code Inspections	
Top-down Development	
Chief Programmer Teams	
Complexity	1.69
Overall Code Complexity	
Complexity of Application	
Complexity of Program Control Flow	
Code Mix	1.47
Proportion of Code classed as Non-Mathematical and I/O Formatting	
Proportion R/T, Interactive, or Time-Critical Code	
Proportion of Code Intended for Delivery	
Misc. Items	* Not Calculated *
Proportion of Data Base Class-Items to 1,000 LOC	
Proportion of Doc. Pages to 1,000 LOC	
Ratio of Staff Size to Project Duration (People/Month)	

Figure 3-26a. IBM/Walston and Felix Marginal Group Productivity Impacts - Listed in Order of Precedence

productivity increase on the order of 70%. Of subsequent interest is the individual contribution from each component comprising the group. To arrive at these individual contribution factors, an equation of the following form had to be solved for:

Marginal Group Productivity Impact = (1+C1) x (1+C2) x .... x (1+Cn) [3.2-b]

where: C1-n represent the marginal contributions of each group's component members

Obviously, this is no trivial task and it appears that the possible component values could take on any one of a wide range of possible values. Fortunately, the solution can be determined due to the implied variable relationships that exist in the basic productivity data. Through the data, the basic equation of the form 3.2-b could be translated to a form described by only one of the variables where the remaining variables are defined through the one variable and a ratio calculated from the original data. This translation, coupled with the facility of a digital computer, greatly simplifies the solution procedure. This basic solution procedure is illustrated mathematically in Figure 3-27. In this example, it is shown how the component marginal contribution rates of the components of the "Structured Techniques" group were determined. The equation of the single variable, A, was solved for on an interactive computer system through a program utilizing an iterative solution technique. With this same procedure, the remaining contribution factors of the groups could be solved for and the results are provided in Figure 3-28. Figure 3-29 summarizes all of the group productivity impacts as well as the component contributions of each

Structured Techniques	Productivity (LSC/PM) No	Yes	% Increase
A) Structured Programming	169	301	78
B) Design and Code Inspections	220	339	54
C) Top-down Development	196	321	64
D) Chief Programmer Teams	219	408	86
Totals -	804	1369	

These four factors affect productivity by $\frac{1369 - 804}{804} = +70\%$

$\Longrightarrow$ 70 % is this Group's Marginal Productivity Impact.

Relationships:

$$(1+A) \times (1+B) \times (1+C) \times (1+D) = 1.7$$

			A	/=/	0.16
A/B = 78/54	B = 54*A/78		B	/=/	0.11
A/C = 78/64	C = 64*A/78		C	/=/	0.13
A/D = 78/86	D = 86*A/78		D	/=/	0.17

Solutions found by:

$$(1+A) \times (1+(54*A/78)) \times (1+(64*A/78)) \times (1+(86*A/78)) \text{ /=/ } 1.70$$

Notes:
- "LSC/PM" means "Lines of Source Code per Person Month"
- " x " symbolizes arithmetic multiplication
- " /=/ " means - "approximately equal to"

Figure 3-27. Sample Calculation of Group Component Contributions

Structured Techniques	Productivity (LSC/PM) No	Yes	% Increase
A) Structured Programming	169	301	78
B) Design and Code Inspections	220	339	54
C) Top-down Development	196	321	64
D) Chief Programmer Teams	219	408	86
Totals -	804	1369	

These four factors affect productivity by $\frac{1369 - 804}{804} = +70\%$

====> 70 % is this Group's Marginal Productivity Impact.

Relationships:

$$(1+A) \times (1+B) \times (1+C) \times (1+D) = 1.7$$

A/B = 78/54	B = 54*A/78	A	/=/	0.16
A/C = 78/64	C = 64*A/78	B	/=/	0.11
A/D = 78/86	D = 86*A/78	C	/=/	0.13
		D	/=/	0.17

Solutions found by:

$$(1+A) \times (1+(54*A/78)) \times (1+(64*A/78)) \times (1+(86*A/78)) \text{ /=/ } 1.70$$

Notes: "LSC/PM" means "Lines of Source Code per Person Month"
" x " symbolizes arithmetic multiplication
" /=/ " means - "approximately equal to"

Figure 3-28. Calculation of Group Component Contributions

Complexity	Productivity (LSC/PM) "Average" >	<	% Increase
A) Overall Code Complexity	185	314	69
B) Complexity of Application	168	349	108
C) Complexity of Program Control Flow	209	289	38
Totals -	562	952	

These three factors affect productivity by $\frac{952 - 562}{562} = + 69\%$

===> 69 % is this Group's Marginal Productivity Impact.

Relationships:

$$(1+A) \times (1+B) \times (1+C) = 1.69$$

A/B = 69/108	B = 108*A/69	A	/=/	Ø.19
A/C = 69/ 38	C = 38*A/69	B	/=/	Ø.29
		C	/=/	Ø.10

Solutions found by:

$$(1+A) \times (1+(108*A/69)) \times (1+(38*A/69)) \quad /=/ \quad 1.69$$

Notes:

"LSC/PM"	means "Lines of Source Code per Person Month"
" x "	symbolizes arithmetic multiplication
" /=/ "	means - "approximately equal to"

Figure 3-28 (Cont.). Calculation of Group Component Contributions

Code Mix	Productivity (LSC/PM) "Relatively:" Ltl	Mch	% Increase
A) Non-math; I/O Formatting	188	267	42
B) Non-Real-time, nor time-critical	203	279	37
C) % Intended for Delivery	159	265	67
Totals -	550	811	

These three factors affect productivity by $\frac{811 - 550}{550} = + 47\%$

$\Longrightarrow$ 47 % is this Group's Marginal Productivity Impact.

Relationships:

$$(1+A) \times (1+B) \times (1+C) = 1.47$$

$$A/B = 42/37 \qquad B = 37*A/42 \qquad A \ /=/ \ 0.12$$
$$A/C = 42/67 \qquad C = 67*A/42 \qquad B \ /=/ \ 0.10$$
$$C \ /=/ \ 0.19$$

Solutions found by:

$$(1+A) \times (1+(37*A/42)) \times (1+(67*A/42)) \ /=/ \ 1.47$$

Notes:
- "LSC/PM" means "Lines of Source Code per Person Month"
- " x " symbolizes arithmetic multiplication
- " /=/ " means - "approximately equal to"

Figure 3-28 (Cont.). Calculation of Group Component Contributions

Utilization	Productivity (LSC/PM) Severe	Minimal	% Increase
A) Program Design Constraints	166	293	77
B) Core Memory Constraints	193	391	103
C) Execution Time Constraints	171	303	77
Totals -	530	987	

These three factors affect productivity by $\frac{987 - 530}{530} = + 86\%$

=====> 86 % is this Group's Marginal Productivity Impact.

Relationships:

$$(1+A) \times (1+B) \times (1+C) = 1.86$$

A/B	= 77/103	B	= 103*A/77	A	/=/	0.21
A/C	= 77/ 77	C	= A	B	/=/	0.27
				C	/=/	0.21

Solutions found by:

$$(1+A) \times (1+(103*A/77)) \times (1+A) \quad /=/ \quad 1.86$$

Notes:
- "LSC/PM" means "Lines of Source Code per Person Month"
- " x " symbolizes arithmetic multiplication
- " /=/ " means - "approximately equal to"

Figure 3-28 (cont.). Calculation of Group Component Contributions

Platform	Productivity (LSC/PM) "Normal" >	<	% Increase
A) Customer Interface Complexity	124	500	303
Degree of User -			
B) Participation in Requirements Spec.	205	291	42
C) Originated Design Changes	196	297	52
D) Experience in Application Area	206	318	54
Totals -	731	1406	

These four factors affect productivity by $\frac{1406 - 731}{731} = + 92\%$

===> 92 % is this Group's Marginal Productivity Impact.

Relationships:

$$(1+A) \times (1+B) \times (1+C) \times (1+D) = 1.92$$

			A	/=/	Ø.34
A/B = 303/42	B = 42*A/303		B	/=/	Ø.12
A/C = 303/52	C = 52*A/303		C	/=/	Ø.13
A/D = 303/54	D = 54*A/303		D	/=/	Ø.13

Solutions found by:

$$(1+A) \times (1+(42*A/303)) \times (1+(52*A/303)) \times (1+(54*A/303)) \; /=/ \; 1.92$$

Notes:
- "LSC/PM" means "Lines of Source Code per Person Month"
- " x " symbolizes arithmetic multiplication
- " /=/ " means - "approximately equal to"

Figure 3-28 (Cont.). Calculation of Group Component Contributions

Resources	Productivity (LSC/PM) Low	High	% Increase
Quality of Currently-available Resources:			
A) Average Personnel Experience	132	410	211
B) % of Prgrmrs who did Design	153	391	156
C) % Utilization of Currently-available Hardware	177	297	68
Degree of Previous Experience:			
D) - with the Computer	146	312	114
E) - with the Programming Language	122	385	216
F) - with a Similar Application	146	410	181
Totals -	731	1406	

These six factors affect productivity by $\frac{1406 - 731}{731} = +173\%$

====> 173 % is this Group's Marginal Productivity Impact.

Relationships:

(1+A) x (1+B) x (1+C) x (1+D) x (1+E) x (1+F) = 2.73

			A	/=/	0.22
A/B = 211/156	B = 156*A/211		B	/=/	0.18
A/C = 211/ 68	C = 68*A/211		C	/=/	0.12
A/D = 211/114	D = 114*A/211		D	/=/	0.15
A/E = 211/216	E = 216*A/211		E	/=/	0.23
A/F = 211/181	F = 181*A/211		F	/=/	0.20

Solutions found by:

(1+A) x (1+(156*A/211)) x (1+(68*A/211)) x (1+(114*A/211))
x (1+(216*A/211)) x (1+(181*A/211)) /=/ 2.73

Notes: "LSC/PM" means "Lines of Source Code per Person Month"
" x " symbolizes arithmetic multiplication
" /=/ " means - "approximately equal to"

Figure 3-28 (Cont.). Calculation of Group Component Contributions

Security	Productivity (LSC/PM) Low	High	% Increase
A) Access Limited to Computer	226	357	58
B) Amount of Open Access to Computer	170	303	78
C) % of Work which is Classified	156	289	85
Totals -	552	949	

These three factors affect productivity by $\frac{949 - 552}{552} = + 72\%$

====> 72 % is this Group's Marginal Productivity Impact.

Relationships:

$$(1+A) \times (1+B) \times (1+C) = 1.72$$

			A	/=/	0.18
A/B = 58/78	B = 78*A/58		B	/=/	0.20
A/C = 58/85	C = 85*A/58		C	/=/	0.21

Solutions found by:

$$(1+A) \times (1+(216*A/211)) \times (1+181*A/211)) \quad /=/ \quad 2.73$$

Notes:

"LSC/PM"	means "Lines of Source Code per Person Month"
" x "	symbolizes arithmetic multiplication
" /=/ "	means - "approximately equal to"

Figure 3-28 (Cont). Calculation of Group Component Contributions

	Marginal and Component Contribution Impacts	
Resources	2.73	
Ave. Personnel Experience and Qualifications		1.22
% Dev'mt. Programmers who Participated in Func. Design Spec.		1.18
% Utilization of Currently-available Hardware		1.12
Degree of Previous Experience with Oper. Computer		1.15
Degree of Previous Experience with Programming Languages		1.23
Degree of Previous Experience with Appl. Size and Complexity		1.20
Platform	1.92	
Customer Interface Complexity		1.34
Degree of User Participation in Req'mts. Def.		1.12
Degree of Customer-Originated Design Changes		1.13
Degree of Customer Experience in Application Area		1.13
Utilization	1.86	
Overall Program Design Constraints		1.21
Core Memory Design Constraints		1.27
Execution Time Design Constraints		1.21
Security	1.72	
Spcl. Req. for Access to Dev'mt. CPU		1.18
Amount of Open Access Time to Dev'mt. CPU		1.20
Classified Security Environment for CPU and 25% of Programs & Data		1.21
Stuctured Techniques	1.70	
Structured Programming		1.16
Design and Code Inspections		1.11
Top-down Development		1.13
Chief Programmer Teams		1.17
Complexity	1.69	
Overall Code Complexity		1.19
Complexity of Application		1.29
Complexity of Program Control Flow		1.10
Code Mix	1.47	
Proportion of Code classed as Non-Mathematical and I/O Formatting		1.12
Proportion R/T, Interactive, or Time-Critical Code		1.10
Proportion of Code Intended for Delivery		1.19
Misc. Items	* Not Calculated *	
Proportion of Data Base Class-Items to 1,000 LOC		
Proportion of Doc. Pages to 1,000 LOC		
Ratio of Staff Size to Project Duration (People/Month)		

Figure 3-29. Summary of IBM/Walston and Felix Group and Component Contribution Impacts on Software Program Development

environmental factor itemized. With the data provided by Aron, Doty et. al., and Walston and Felix, general guidelines governing the application of scaled system techniques can be presented.

3.3.3 Generalized Guidelines for Scaling Systems

After determining the Walston and Felix marginal group productivity impacts listed in Figure 3-26 through the procedures already described, the groups could then be ranked according to their potential impact upon a software development effort. This has already been done, as may have been noticed, in Figure 3-26a. Consequently, the summary format of Figure 3-29 adheres to the same ranking. The significance of this ranking to the system planner is the relative importance of each environmental attribute to the construction of cost effective software systems. The Doty factors were ranked in a similar manner and are presented in Figure 3-30. Of importance to the potential practitioner of the scaled system development methodology are the priorities that personnel experience, use of available hardware, and establishment of operational and functional requirements hold in the determination of an average productivity estimate reflecting the software development effort. From the data presented, quantitative guidelines such as those that follow may be observed:

3.3.3.1 Personnel Experience

Scaled systems benefit environments characterized by inexperienced technical staffs and/or technical staffs faced with the challenge of developing state-of-the-art or otherwise unique systems. From the Walston and Felix data, the experience an otherwise inexperienced technical staff gains from a scaled system implementation can be expected

Factor	Effort Increase (Maximum)
CPU Time Constraints	132%
Concurrent Software and Hardware Development	122%
Development CPU Different from Target CPU	122%
Detailed Definition of Operational Requirements	100%
First Software Developed on CPU	92%
Development at More than One Site	75%
Real-Time Operation	67%
Limited Programmer Access to Computer	50%
Developer Using Computer at Another Facility	43%
CPU Memory Constraints	43%
Special Display	43%
Development at Operational Site	39%
Changing Operational Requirements	5%
Time-Share, Interactive Development [decreases effort -]	21%

Note: Percentage figures come from maximum factors for the particular environmental attributes listed in Figure 3-23.

Figure 3-30. Doty Factors Ranked in Order of Adverse Impact on Software Development

to increase productivity in the unscaling effort by a factor of approximately 173%. This translates to approximately three times the average productivity, or approximately one-third the effort, otherwise expected of an inexperienced staff. In contrast, the Doty factors reflect that up to a 92% increase may be achieved based upon the staff gaining familiarity with the computer equipment alone. Barry Boehm, in Practical Strategies for Developing Large Scale Software Systems, quantified the resulting benefits of an experienced staff to be on the order of 150-200% [ref. 9]. Despite the various sources, all of this data appears to be relatively consistent. This is reasonable since the performance of people should not be expected to change much over time.

3.3.3.2 Customer Environment - The Platform

Software system implementors faced with customer environments characterized by such factors as complex customer interface channels and procedures, customer uncertainty and inexperience regarding operational and functional requirements, and the potentially numerous customer-originated design changes which subsequently result from inadequate requirements specification can greatly benefit from the scaled systems approach.

Dr. Daniel Teichroew, a professor of industrial and operations research in charge of the ISDOS (Information Systems Design and Organization System) project at the University of Michigan who is also credited with the development of the Problem Statement Language/Problem Statement Analyzer (PSL/PSA), points out that the front-end stage of an implementation, where the requirements and high-level design are specified, is the pitfall of past failures. In his words, "this often

overlooked phase is where most of the problems and potential payoffs lie (in software development projects)" [ref. 20].

3.3.3.3 Firm Requirements Specifications

An appropriately scaled system can provide difficult customer environments with the experience and knowledge necessary for the determination of the precise customer needs.

The importance of establishing the user's needs, in any effort, cannot be overstated. Two adverse situations may develop in the absence of a firm specification of the user's needs: (a) the developer implements a system based on an incomplete or incorrect specification, and the system is rejected by the users, or (b) the developer continually changes the system design based on conflicting direction from the user. The first situation results in a system that fails to meet performance and functionality requirements, while the second situation greatly increases system cost and development time, and also runs the risk of entering a never-ending change cycle in which the system is never actually completed.

In most cases, failure to have a firm user specification is not the fault of the user, but is rather due to the user having incomplete information as to exactly what is feasible and practical with an automated system. A scaled system built at a fraction of full-scale system cost can be used to demonstrate exactly what capabilities are available to the user, as well as to give the user an idea of how he will interface with the system and what procedures must be developed. Based upon his experience with a scaled system, the user will then be able to clearly specify his requirements for the full-scale system.

The potential benefits resulting from such use of scaled systems are large. Design changes as a result of requirements errors have been documented [ref. 9,10] as being 50 to 400 times more expensive in comparison to front-end design changes (before actual implementation commences). From the Walston and Felix data, the potential for benefits in the unscaling effort are on the order of 92% - nearly twice the productivity, or, conversely, half the effort. The Doty factor for changing operational requirements seems a contradiction to other research, as it assesses a mere 5% penalty for changing requirements. While this low value cannot be explained, it also cannot be corroborated with any other research examined under this effort.

3.3.3.4 Hardware Choice

In the event the scaled system can provide information facilitating the choice of hardware such that ultimate full-scale system speed and memory constraints may be more readily complied with, the Walston and Felix data suggests a potential 86% increase in productivity applying to the unscaling implementation effort.

3.3.3.5 Secure Environments

Planners of intelligence systems may additionally be interested in the fourth most important cost driver identified in the Walston and Felix data, that of secure, or classified, operating and/or development environments. The potential for benefits resulting from implementing an otherwise classified system in an unclassified environment through the use of dummy test data and other techniques approaches 72%.

-Such examples are provided to illustrate to system planners the general method of determining guidelines through analysis of factors

pertinent to system costs. Consequently, they may utilize this data or make use of new data as it becomes available to determine the benefits and potentials existing in the scaled system methodology as it may apply to any particular software development endeavor. The potential benefits realized through experience and general system information are paramount considerations in the application of the scaled systems approach.

3.3.4 Use of the Individual Walston and Felix Group Component Rates

It is reasonable that the individual Walston and Felix component contribution rates can also be interpreted in an analogous manner to the application of the generalized group productivity impacts. For example:

3.3.4.1 State-of-the-Art Hardware

In the event that a system requires state-of-the-art hardware not yet available, the Walston and Felix data predicts the scaled system implementation effort can benefit by a 12% increase in productivity through the use of currently-available hardware. This could include special, reusable hardware specifically programmed and configured for scaled system implementations. In contrast, the Doty factors quantify the savings realized through the elimination of the environmental factor of concurrent hardware development with the software effort to be on the order of 112%. Such hardware substitution is greatly facilitated through the increasing use of higher-order (HOL) programming languages and, with the approaching standardization within the DoD environment to ADA, scaled systems will be more readily transportable for later enhancement regardless of what hardware is used to implement them.

3.3.5 Degrees of Scaling Freedom

A major point made in "Scaled Systems Cost Effectiveness", a technical memorandom submitted earlier under this research effort (reprinted in Appendix A), was the application of cost/benefit, or "break-even", analytical techniques to assessing the cost trade-offs of using scaled systems. Such techniques are very useful to practitioners of the scaled systems methodology in determining the cost/effort/schedule feasibility of applying scaled systems techniques. The primary objective of break-even analysis is to determine an overall system scale factor, or ratio, such that the total estimated development cost of both the scaled system and its full-scale counterpart "breaks even" with the estimated cost of a full-scale implementation effort without the benefit of a scaled system. This break-even point is important because it reveals that a scaled system built on a smaller scale relative to the break-even scale factor should result in overall project savings; conversely, if the break-even scale factor cannot be achieved, then total project costs may well be expected to exceed the cost of a traditional implementation approach. Of course, this additional estimated cost is subject to justification based upon such factors as a reduction in total project risk, the achievement of design or performance goals, the development of a user-friendly interface, or some other applicable success-oriented criteria.

In "Scaled Systems Cost Effectiveness", break-even points were determined through parametric analysis using an interactive software cost estimation model. In that study, the break-even points were found to be cost model dependent and displayed a static relationship with system size across a wide range of system sizes. This static relationship is

attributed to the cost estimation relationships (CERs) internal to the model and may or may not hold in actual practice.

Through subsequent research, it was determined that break-even points could be directly calculated from productivity coefficients like the Doty and the Walston and Felix environmental factors. This calculation consists of simply subtracting the inverse of the factor from the value of one. The resulting value quantifies two things. First, it maintains the relative measure of importance the environmental factor originally quantified to its potential impact on a software development effort. Secondly, and most important to scaled system cost/benefit analysis, the resulting factor quantifies a break-even value for determining potential scaled system cost effectiveness. This break-even value represents the break-even scale factor for an implementation effort characterized by a scaled system bearing the burden of the negative impact of the environmental factor while the full-scale counterpart realizes the benefits resulting from removal of the negative burden. Because these factors rank environmental factors by overall system break-even scale factors, they inform the scaled system practitioner of the freedom of constraints he has in the construction of the scaled system providing the benefits of the scaled system include removal of the negative impacts arising through the particular environmental factor. This freedom factor reflects the constraints placed upon the scaled system in terms of size, effort, and cost. Subsequently, these freedom factors have been termed "Degrees of Scaling Freedom". The Walston and Felix-derived factors are listed in Figure 3-31 while the corresponding Doty-derived factors appear in Figure 3-32.

	Break-Even Scale Factor
Resources	63%
Ave. Personnel Experience and Qualifications	18%
% Dev'mt. Programmers who Participated in Func. Design Spec.	15%
% Utilization of Currently-available Hardware	11%
Degree of Previous Experience with Oper. Computer	13%
Degree of Previous Experience with Programming Languages	19%
Degree of Previous Experience with Appl. Size and Complexity	17%
Platform	48%
Customer Interface Complexity	25%
Degree of User Participation in Req'mts. Def.	11%
Degree of Customer-Originated Design Changes	12%
Degree of Customer Experience in Application Area	12%
Utilization	46%
Overall Program Design Constraints	17%
Core Memory Design Constraints	21%
Execution Time Design Constraints	17%
Security	42%
Spcl. Req. for Access to Dev'mt. CPU	15%
Amount of Open Access Time to Dev'mt. CPU	17%
Classified Security Environment for CPU and 25% of Programs & Data	17%
Stuctured Techniques	41%
Structured Programming	14%
Design and Code Inspections	10%
Top-down Development	12%
Chief Programmer Teams	15%
Complexity	41%
Overall Code Complexity	16%
Complexity of Application	22%
Complexity of Program Control Flow	9%
Code Mix	32%
Proportion of Code classed as Non-Mathematical and I/O Formatting	11%
Proportion R/T, Interactive, or Time-Critical Code	9%
Proportion of Code Intended for Delivery	16%
Misc. Items	* Not Calculated *
Proportion of Data Base Class-Items to 1,000 LOC	
Proportion of Doc. Pages to 1,000 LOC	
Ratio of Staff Size to Project Duration (People/Month)	

Figure 3-31. Degrees of Scaling Freedom as Derived from the IBM/Walston and Felix Data

Factor	Break-Even Scale Factor
CPU Time Constraints	57%
Concurrent Software and Hardware Development	55%
Development CPU Different from Target CPU	55%
Detailed Definition of Operational Requirements	50%
First Software Developed on CPU	48%
Development at More than One Site	43%
Real-Time Operation	40%
Limited Programmer Access to Computer	33%
Developer Using Computer at Another Facility	30%
CPU Memory Constraints	30%
Special Display	30%
Development at Operational Site	28%
Changing Operational Requirements	5%
Time-Share, Interactive Development [decreases effort]	N/A

Note: Break-even scale factors come from maximum factors for the particular environmental attributes listed in Figure 3-23.

Figure 3-32. Degrees of Scaling Freedom as Derived from the Doty and Associates Data

A scenario for general application of these degrees of scaling freedom follows:

The system planners scan the lists of environmental factors and determine which ones characterize the particular development environment under scrutiny. These factors normally cause adverse impacts upon a development effort and, hence, will have much the same impact on the scaled effort. However, since the scaled effort is not as great as the full-scale attempt, the absolute value of the penalties imposed by the adverse environmental factors are much less for the scaled effort. Ostensively, these negative impacts will be removed and thus not affect the up-scaling effort due to lessons learned through the scaled effort. The overall savings resulting from this process contribute toward the cost of the scaled effort and, possibly, to total project savings. The question arises - how small must the scaled system be in order to achieve cost savings? Obviously, it is hoped that the size of the scaled system will not be so constrained that its operational value and predictive ability are minimized. The answer to the question lies in the degree of scaling freedom values like those listed in Figures 3-31 and 3-32. As a gross surrogate, the system planner may initially simply use the maximum of the applicable values as the break-even system scale factor, relying upon the others to "back-up" this estimate and to add to the measure of confidence in its use. Adding the break-even scale factors together is not recommended as such a technique would most probably tend to produce an overly-optimistic break-even scale factor estimate. Root mean square analysis may be applicable to the situation. To determine the root mean square, the analyst calculates the square root of the sum of the squares

of all the degrees of freedom values which correspond to the environmental attributes existing in the proposed effort. Mathematically, the root mean square calculation looks like this:

$$\text{Break-even Scale Factor} = \sqrt{\sum_{i=1}^{\#\text{ of applicable environmental factors}} \text{Environmental factor}_i \uparrow 2}$$

The validation of such an analytical technique is beyond the scope of this research and must be left to future research of the application of the scaled system methodology, as presented here, to actual systems. At least these degree of scaling freedom factors are based upon actual, credible empirical data. Again, a very conservative break-even scale factor for a potential scaled system development effort can be obtained from the maximum environmental factor degree of freedom value listed in the tables.

The Scaled System Cost Effectiveness study determined the range of the degree of scaling freedom values to vary from 10 to 50%. This means that, based upon that study, scaled systems of 10-50% overall scale can be expected to achieve cost savings. The variation can be attributed to the application type and environmental factors present. These results are not altogether inconsistent with the IBM/Walston and Felix-derived degrees of freedom, which range from 9-63%, nor the Doty-derived degrees of freedom, which range from 5-57%.

Based upon this study of empirical data, a general rule of thumb can be derived: In order to achieve cost effectiveness, a scaled system would most probably have to be scaled by at least 50%; however, in order

for the scaled system to retain any predictive value, the scaled system should not be scaled to less than 10%. Obviously, the verification of this general rule can be achieved only through actual practice of the scaled system methodology in a very controlled and carefully documented manner. The resulting analysis of the data thus provided will certainly contribute toward bettering the methodology and enhancing these predictive measures. It is encouraging that, at this point, the evidence suggests scaled systems built to as large as 50% of the actual system can indeed provide cost savings as well as ensuring the production of quality software systems.

3.4 Cost Benefits

The estimation of implementation-dependent cost benefits resulting from the use of scaled systems techniques relies heavily upon the capabilities of current cost estimation models and methodologies as well as the subjective analysis performed by the experienced system planner. A general familiarity with the operation and capabilities of currently available software cost estimation models is therefore required of the potential scaled system practitioner. Accordingly, a general discussion of current cost estimation model state-of-the-art is provided in this section to familiarize the reader with these models and their use. As a conclusion of this section and report, a case study of an actual intelligence system is presented to serve as a model for subsequent scaled system feasibility/cost benefit analysis.

3.4.1 Life-Cycle Cost Estimation Models

Estimation of the costs and schedule for software development is crucial to accomplishing effective planning, budgeting, and evaluation

activities within an organization. These are the activities that in the world of software project management have long been documented as historical problem areas. Optimistic cost projections along with gross errors have contributed to severe budget and schedule overruns.

The importance of software project life-cycle cost models has long been recognized in the process of making viable software cost estimates and the efficient allocation of resources. This does not mean that total reliance should be placed upon the cost model to singly accomplish cost estimating. It must be accepted in the context of a comprehensive cost estimation strategy where the cost model is viewed as a tool for the competent cost analyst. Its value is derived from the imposition of a disciplined and structured framework that compels the analyst to consider and take into account all significant factors influencing software development costs. The software life-cycle cost model is a valuable tool that can account for complex nonlinear relationships between seemingly random data through use of automated mathematical and statistical analytical techniques.

3.4.1.1 Cost Estimation Model Methodology

In general, cost model operation involves calibration to historical experience, input parameter determination, model operation and cost analysis/presentation, and risk assessment.

(1) Calibration

Representation of the developmental environment is the single most important factor determining the model's applicability to projecting cost behavior. The cost model must be either carefully designed to model cost behavior within that environment or have the capability to adapt

itself to reflect the cost characteristics of any specified environment. Consequently, the commercially available cost models are supplied with the facility to be calibrated to a specific environment. This calibration process is crucial to the ability of the model to project cost behavior within a particular environment.

The objective of the calibration process is to tailor the model through the use of an organization's historical cost data so that the model's predictive ability, within the organization, is enhanced. Most models have special functions to determine the variables for this purpose and make them available to the user, in the form of an input parameter, for subsequent use by the model. This input parameter is a global descriptor, reflecting the professional quality and problem-solving capabilities of the organization's technical and administrative staffs.

Two commercially available cost estimation models, PRICE S and SLIM, each have these inputs. One reason such variables are made available to the user is that past development conditions may no longer hold. The user may subsequently find it necessary to adjust these variables in order to achieve more realistic cost estimates. Examples of such conditions include the adoption of newer, more up-to-date structured development practices, the acquisition of more powerful development tools and facilities, or an increase in the skill level of the organization's personnel resulting from prior experience (the converse of this condition, the decrease in skill level due to personnel turnover and new hires, is also possible).

(2) Input Parameter Determination

There are so many factors that can potentially effect program development costs that it is virtually impossible (and impractical) for a cost model to attempt to deal with them all on an individual basis. This problem has been tackled by grouping related factors and representing each category by a generalized model input parameter. This is where structured systems thinking is required - resulting in the disaggregation of a large, ambiguous task, into a structured decomposition of several smaller, more manageable tasks for cost estimation. It is therefore necessary for the cost analysis team to aggregrate the effects of all related factors so that their combined effects may be synthesized into the appropriate mix of model input parameters.

The schema for input parameter estimation includes such all-encompassing project considerations as staff capabilities, product attributes, application requirements, environmental factors, development practices, and management policies. Intelligence systems built to military specifications would include, as an example, general provisions for product attributes such as real-time operation, modularity, and strict documentation standards for usability and maintainablilty; application requirements for testability, quality assurance, and reliability; environmental factors such as secure developmental and operating facilities; development practices such as structured design walk-throughs and close progress tracking; and management policies in regards to staffing and the scheduling of project milestones.

(3) Model Operation - Output Analysis, Presentation

Once the cost analyst team has collected, analyzed, and synthesized all of the pertinent cost data relevant to the proposed

development effort into the appropriate model input parameters (and calibrated the model, if sufficient historical data is available), they are ready to use that data to exercise the model. The model's output consists of milestone schedules, staff-loading profile charts, and some measure of cost expressed either in terms of personnel effort or dollars. In addition, the model may break down the expenditure estimate by labor category such as technical, managerial, coding, documentation, etc. and/or functional category such as design, code, test and integration, maintenance, etc. In the event that the model encounters a set of input data which is inconsistent with the formulated guidelines, it will also produce the appropriate warning or error messages.

(4) Estimate Risk Assessment

One aspect of the cost estimate which the particular model may address is the measure of uncertainty, or risk, associated with the cost estimate. This is usually either a standard statistically-derived measure upon the estimate, such as a root mean square (standard deviation) value, or the preparation of a sensitivity profile obtained through parametric analysis of the input parameters. In order to realize the full meaning and value of such risk measures the assumptions and method by which they are derived must be understood by the cost analyst.

3.4.1.2 Cost Estimating Considerations

An important consideration often overlooked or misunderstood is that cost estimation models, in and of themselves, are not a panacea to the general problems inherent in software cost estimating. This has been documented by user groups that have actual experience with cost estimation and evaluation methodologies. These groups stress the value

and importance of juxtaposing the results obtained from the automated cost estimation models with the sound judgement and professional experience of the available technical staff members to produce credible and realistic cost estimates. This is necessary because the cost models quantify past experience. Cost projections based upon such historical data include a certain degree of risk because of the advances which are occurring in the software industry.

Another source of error in the models is their extreme sensitivity to relatively small adjustments in their input parameter mix. This is because the underlying software cost relationships are characterized by complex exponential functions determined by the intricate interrelationships of the various input parameters themselves. This problem is particularly acute when using input parameters that fall outside the ranges for which the model was calibrated.

Solutions to these problems include fine-tuning of the input parameter mix to match preconceived cost targets as well as assessing the already mentioned existence of wide variations (risk) in the estimates. These variations raise an important philosophical point: that the dynamics of software cost estimating are such that obtaining high accuracy in the point estimates is neither possible nor desirable due to the calculational variation which is present.

The utility of the costing model lies in the structure and discipline imposed upon the costing process. The reliability of the resulting estimate is dependent on the assumptions which produced it. All estimates must be scutinized by experienced cost analysts to ensure that the results, along with the underlying assumptions, are reasonable.

3.4.1.3 Assessment of Cost Model State-of-the-Art

Growing acceptance of life-cycle models arises not only because of their potential to serve management in the planning, programming, monitoring, and evaluation of software production efforts (through the provision of schedules, manpower allocation profiles, and cost estimates), but also because of their merits in creating a structured and disciplined approach to the estimation and evaluation of cost estimates to serve decision-makers' needs. Nevertheless, attention to the capabilities, limitations, characteristics, and purpose of software life-cycle cost models must be fixed in the minds of those who use, as well as those who develop them.

Note the current state of development of the estimating technology. Software life-cycle cost models are in an infant stage of their product life-cycle. They are still growing in acceptance and popularity. Advances are being made in their underlying theoretical formulation as well as in refinement of their operational and functional characteristics.

The skepticism of those who consider these models to be expensive frills is not altogether unfounded. The cost of the commercially available models is artificially inflated due to the profits required by the vendors to recoup their large investment in research and development. However, as in all newly marketed technologies, it is not unrealistic to expect that, as economies of scale and competitive market forces come into play, the prices should decline.

As for the life-cycle models themselves, we can, for convenience, classify them into two general groups: commercial general purpose and

academic special purpose.

(1) Commercial General Purpose Cost Models

Under the category of commercial general purpose models, we find two popular models. One is RCA's PRICE S, which is actually a member of a family of three related cost estimation models which also includes PRICE, a hardware manufacturing cost estimation model, and PRICE SL, a software life-cycle cost estimation model. The other is SLIM, a product available for lease from Quantitative Software Management, Inc., headed by Lawrence Putnam. Mr. Putnam is credited with making, and publicizing, significant advances in the area of the theory of software costing and estimation.

Although not commercially available, a new integrated approach to cost estimating and evaluation occurs in the form of a system which utilizes both the PRICE S and a modified version of the Putnam model to evaluate cost proposals at the Space and Missile Systems Organization's (SAMSO) Software Programs Office (SPO) at Los Angeles, California. The system is implemented on a Hewlett-Packard Series 3000 and supports generalized pre-processing interpretation of standardized input formats for subsequent input to both models as well as graphics-oriented, post-processing facilities for both of the model's outputs. Additionally, the system has an integral data base management system and a financially-oriented report generator.

(2) Academic/Special Purpose Models

Widely discussed in the research literature and at various conference and workshop proceedings is a collection of software estimation models which result principally from special purpose and

academic pursuits. These models have a limited range of applicability since they reflect specific environments, a limited scope of applications, and/or products of similar sizes and attributes. Thus, they are not considered general purpose cost estimation models.

Perhaps the most frequently referenced article on the subject of quantifying software development productivity rates and estimation ratios is that of IBM's Walston and Felix, which first appeared in an IBM technical journal in 1977 [ref. 24]. In the literature we additionally find many papers describing and commenting on the theories of cost estimation set forth by Mr. Putnam, which is principally an extension of the work performed by still another IBM researcher, Peter Norden [ref. 15]. From efforts expended by and under the direction of Victor Basili, [refs. 3, 4, 5, & 6], of the Computer Sciences Department at the University of Maryland, comes a rich proliferation of articles, dissertations, and research findings encompassing a vast array of software engineering topics, including cost and cost-related modeling.

The government itself is active in the areas of software engineering and development of cost estimating techniques with research grant activity through organizations such as the NASA/Goddard Space Flight Center, [refs. 2, & 4], the Rome Air Development Center (RADC), [ref. 7], and various educational institutions, including the University of Maryland. This activity is highlighted by the published findings of Putnam [ref. 17-21] and Doty Associates, Guidelines for Improved Software Cost Estimating, [ref. 7], as well as through the personal and professional-level contributions made by cost analysts such as William Lasher [ref. 9] of the already mentioned SAMSO SPO.

It is upon such research efforts and activity that INCO has based the development of its own version of a software life-cycle cost estimation model. This software life-cycle cost estimation model was targeted for in-house operation on low-cost microprocessor hardware.

3.4.2 Estimation of Scaling Benefits

In the context of scaled system development, cost estimation must take into account the differences that exist between the scaling and up-scaling environments. Certainly, there is a host of factors to take into consideration. Generally, the scaling environment will resemble that of most other developments not using the scaled systems approach. Potential differences in the scaling environment might include the benefits of such scaled system facilities as special low-cost, general-purpose hardware and software development tools specifically tailored to scaled systems development and a greater degree of technical user orientation. Aside from such advantageous environmental niceties, however, the scaled system development environment will most generally be subject to the same negative environmental impacts as most start-from-scratch development efforts.

The positive impacts will be realized in the unscaling environment. The unscaling environment will be much more conducive to system development due to the lessons learned through the scaling effort regardless of the degree of success obtained during the scaled effort. Characteristics of unscaling environments include firm specifications of user-oriented functional, operational, and design requirements. In addition, the unscaling effort will probably benefit from any inventory of design documentation and source code accumulated through the scaling

effort. Also, there should be an optimal hardware configuration chosen to precisely match the needs of the particular application based upon the experience of the scaled system development. By the time up-scaling takes place, the task breakdowns and schedules will have been well defined and laid out and all the parties involved will have a strong, unified concept of the end result and will be in agreement as to what the common goals of the system are. In short, an optimal environment will have been developed complete with fully-detailed descriptions of the tasks at hand and the product to develop -- all unencumbered by the greatest proponents of development project risk and schedule slippage - functional and technical uncertainty.

To quantify the impacts, benefits, and trade-offs inherent in these environments, factors such as those supplied by the Doty and Walston and Felix research efforts, as well as this effort, are available. Some cost estimation models can specifically account for these factors, others will have to be adjusted or modified to do so. At the highest level, these factors can augment experienced technical manager's subjective assessments and their resulting estimations of a project's cost, schedule, and risk.

3.4.3 A Case Study for Scaled Systems

In terms of examining actual intelligence systems which could benefit from application of the scaled systems development approach, this research had the opportunity to explore the possibilities of one such system. Fortunately, this system is somewhat unique in that a corresponding scaled version exists and is operational. Although this scaled version was developed after the full-scale implementation, its

existence provides a "hands-on" feel for examining the application of analytical techniques suitable to the scaled systems methodology. This seemingly "reversed-scaling" approach to system development resulted from different motivations for this particular scaled system and this must be kept in mind so as not to introduce any bias in our case study. As stated, this scaled system realized benefits from the full-scale implementation which was operational, or at least semi-operational, at the time the scaled implementation took place. This scaled system, the Indications and Warning Training System (IWTS), was motivated by the need for a simulator to train intelligence analysts on how to operate an interactive terminal-oriented communications, command, and control intelligence system - the NMIC.

3.4.3.1 Background

After the NMIC achieved an initial operating capability in 1978, INCO, INC. responded to the need for user training with a low-cost, stand-alone analyst training system complete with its own special-purpose hardware. As stated, this system is referred to as the IWTS and it was delivered in early 1979. Because this training system appears to the user as the "real" NMIC system and it models 100% of the full-scale system's functional and operational characteristics, we take the view that it is "the scaled system that could have been".

3.4.3.2 System Development Data Collection

As the implementor of the IWTS, INCO had sufficient data readily accessable concerning its development environment, operational characteristics, and required development effort and costs. Unfortunately, the same was not entirely true of the actual NMIC system;

however, due to INCO's past and present involvement with the design, development, maintenance, and enhancement of the NMIC system as well the completion of the NMIC Functional Analysis/Enhancement Study [ref. 14], adequate data was available to prepare this case study.

3.4.3.3 The NMIC System

Upon initial familiarization with the NMIC system, it appeared to be a nightmare for system implementors, having nearly all of the adverse characteristics possible of a state-of-the-art intelligence system. Of course, many of these characteristics incrementally complicate the development of such a system and drive development cost, schedule, effort, and risk accordingly higher - a perfect candidate for application of the scaled system development approach.

Through this research effort, many of these characteristics could be identified and classified as either new hardware design and development, new software system design concepts, or intelligence system dependent factors.

As for new hardware, the NMIC boasted a wide assortment of state-of-the-art hardware concepts. It was envisioned to be a clustered network of multiple minicomputers interconnected by a new bus technology and system architectural concept. It was to communicate with its users through a totally new, concurrently developed and programmed, intelligent dual-screen full function video/graphics terminal. There were to be fifty such terminals dispersed geographically. Being a message and communications system, it was to process a number of real-time inputs and outputs.

As for new systems design concepts, the NMIC incorporated several new ideas based upon motivations for system reliability, automatic system error and failure recovery, user flexibility, and high-level access to a number of other existing intelligence systems and networks. The basic internal functions of the system were to be distributed throughout the minicomputer network; hence, reliability and functionality were enhanced through the provision of each minicomputer to perform its corresponding function. If a minicomputer failed, only that function would be incapacitated - but what about the recovery of that function? In anticipation of such an event, the NMIC system originally incorporated a system design concept of "fall-back and recovery", whereby another member of the computer network would recover the function lost by the failed computer. Since real-time processing was an integral part of the system, provisions for the design and coding of much time-critical (assembler language) code had to be made.

Additionally, the NMIC system was confronted with a variety of factors typical of an intelligence system. First, the development and operational environments were characteristically security sensitive. The members of the developmental and operational staffs were thus required to have or obtain the appropriate security clearances in order to work on and have access to the system. Such an environment increases development costs because it requires additional controls to be imposed upon the development facilities and personnel by way of locks, guards, logs, etc. and because personnel may not be available, may have to be unproductively employed while waiting for their clearances, or may have to be selected on the basis of clearance rather than skills. While such an environment

is necessary in order for a system and its staff to handle classified material, it creates the opprotunity for scaling of developmental cost by developing technical concepts in a lower-cost, non-secure environment. For example, the "fall-back and recovery" concepts and algorithms for the NMIC system have been developed outside the secured environment. Also, due to its potential importance to the national security, the system development required the most stringent of management, design, documentation, and configuration control practices as well as a large degree of operational functionality, reliability, and robustness.

The varying degrees of success the NMIC achieved in meeting all of its functional and design goals are largely a matter of record and not of great importance to this case study. What is of concern here is the measurement of: (1) the negative productivity impacts the NMIC development sustained in the face of its developmental, environmental, and operational obstacles, and (2) the benefits a scaled prototype might have contributed to the achievement of the NMIC's overall objectives in terms of functionality, budget, and schedule.

3.4.3.4 The IWTS System

The NMIC did not have the luxury of a cost-effective scaled prototype version to facilitate its specification, design, and development. If it did, however, the resulting scaled system would most probably have closely resembled the INCO IWTS trainer. A requirement of the IWTS trainer was to fully provide and support all of the NMIC system's functional and operational features at the user terminal level. The IWTS was developed on low-cost, stand-alone, microprocessor based hardware and, as such, provided very little in the way of the actual

full-scale system's capabilities to receive, rout, and send "real" messages. Such processes were emulated, however, through a pre-stored set of messages reflecting general scenarios of communications that the potential NMIC user would normally encounter. The end result was that the user operating the IWTS terminal has virtually no idea that he is actually using the trainer; but rather has the impression that he is logged-on to the full-scale NMIC system. A diagram of the IWTS design is illustrated in Figure 3-33.

Of importance to this case study is the measurement of the degree of scale the IWTS trainer achieved relative to the NMIC system, its relative cost, and potential predictive ability. The amount of potential savings the NMIC could have actually realized through the use of a system like the IWTS to serve as a scaled prototype development testbed is however basically a matter of conjecture as such an estimate could only be based upon hindsight.

Hardware scale factor is perhaps least difficult to compute. Here, dollar costs are used as the metric since they are most readily available, tangible, and understandable in nature as opposed to some ambiguous measure such as hardware system capacity or power. Relying on a figure obtained from the results of INCO's DIIS FA/ES final study report, the estimated hardware cost of the current NMIC configuration approximates $3.5 million. This figure does not, however, reflect final, enhanced configuration hardware costs of roughly $9 million. These costs reflect the use of current state-of-the-art minicomputers and their associated high-speed I/O peripherals, as well as the U-1652 dual-screen terminal. In contrast, the IWTS, while using the same user terminal,

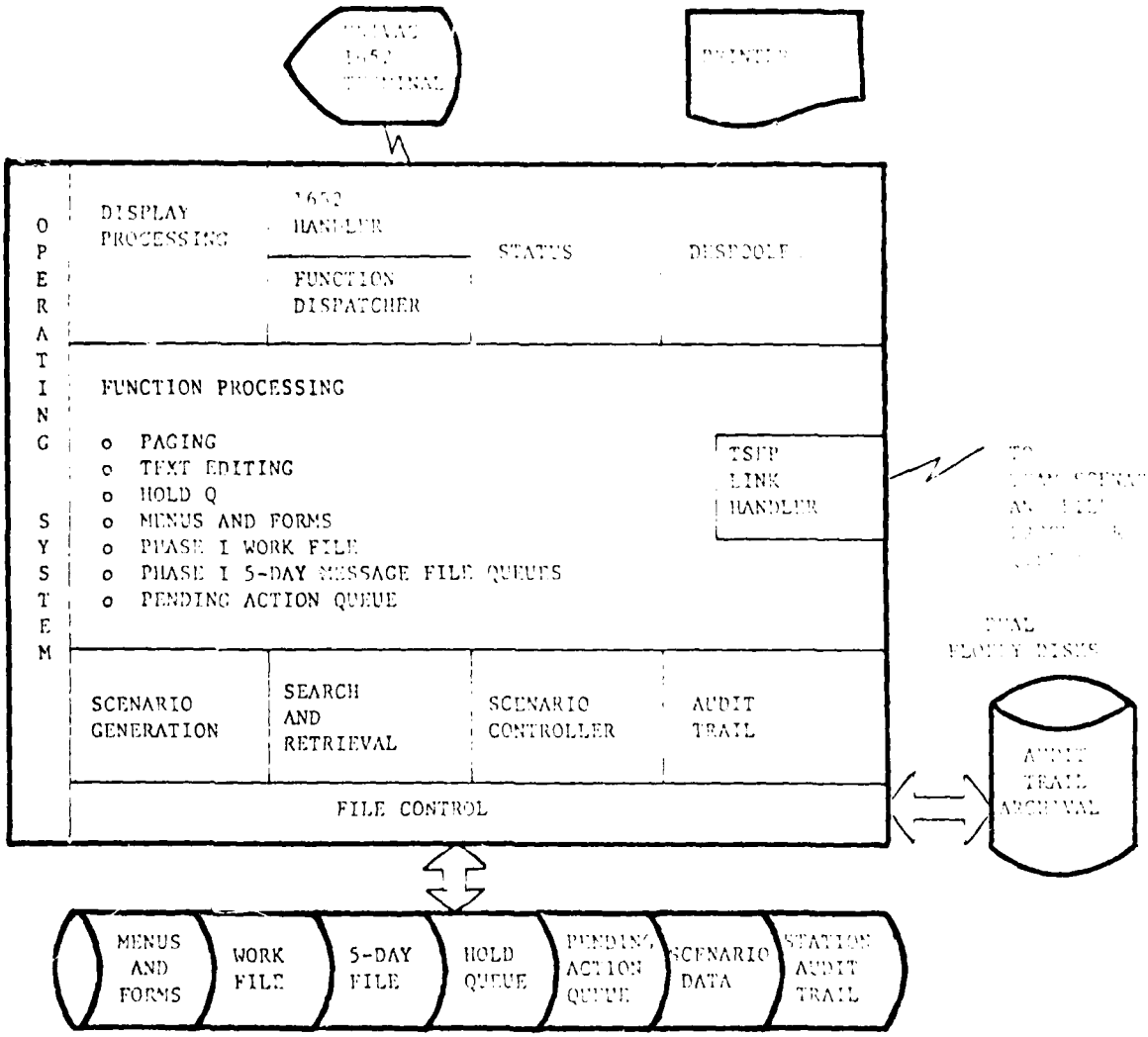


Figure 3-33. IWTS as a Microcosm of the NMIC System

makes use of relatively lower-cost microprocessor-based hardware. The IWTS basic hardware configuration cost is in the neighborhood of $50K, with most of the cost being attributed to the cost and availability of the U-1652 dual-screen, "TEMPEST"-certified (electronic emanation) terminal. Using the cost of the current NMIC configuration, the resulting hardware cost scale factor is computed through:

$$\frac{\$50,000}{\$3,500,000} \sim 1\ 1/2\ \%$$ Scale Factor for Hardware Cost

Of course, a scaled development approach would have required the purchase of both hardware configurations, increasing total project costs. A consideration would be the security classification of the micro hardware in order to implement some of the classified features of the final system; failure of the hardware to obtain the prerequisite certifications would necessitate the use of software emulation techniques to implement similar, non-classified versions of the functions.

Software sizing proves to be a much more difficult task. For one thing, the word sizes and instruction lengths of minicomputers differ from microcomputers. This situation presents a number of problems. First, assembler source language statement counts and required core memory sizes are not directly comparable. Secondly, the statement count for the micro is, in all probability, inflated as compared to the mini since more instructions are required to perform similar tasks because the relative computational and logical power of the micro is not as great as that of the mini. The microcomputer programmer finds himself generating several primitive instructions on the micro where a single instruction

might accomplish the same result on the more powerful mini.

The full-scale NMIC system's module sizes were used as the standard basis on which to compare the different software sizes of the two systems. In this way, it was hoped the problems of the two system's differing language dialects could be resolved as well as maintaining some degree of consistency in the software sizing analysis. Subsequently, it was determined that much of the IWTS trainer's processing was performed in the NMIC's "USS", or User Support Subsystem. Accordingly, the estimated size of the USS served as a surrogate value and a consistency check on the trainer's estimated size. Estimated sizes are used in lieu of the prohibitively long process of actually counting source statements from listings and because of the problems inherent in adding sizes of differing computer language dialects, such as assembler and HOL. In the case of the NMIC, the assembly language used in the bulk of the system's modules was PDP MACRO-11 and FORTRAN was used where an HOL was applicable. Intel 8080 assembler was used for the IWTS with HOL applications programmed in BASIC.

The end result of the sizing analysis resulted in the sizes of the two systems being set to 15,000 source lines of code (SLOC) for the IWTS and 80,000 for the NMIC. These sizes resulted primarily from assessments made by INCO technical staff members who participated in the development of the IWTS and NMIC systems as well as those who performed the DIIS FA/ES study. The size for the IWTS was intentionally set pessimistically high and the size for the NMIC optimistically low in order to avoid any bias in the resulting analysis. The results of the computation for software size scale factor follow:

15,000

——— ~ 20 % Scale for Software Size

80,000

The tracking of actual development effort frequently escapes the ability of most organizations as the means of data collection is usually not present and the figures get absorbed in the aggregation of total labor hours expended throughout the entity. For the analysis of development effort scale, the Doty cost model was used to estimate the amount of effort expended on the NMIC because the actual figure was unobtainable. Data provided by managers of the IWTS project was used as the effort measure for that system. To maintain consistency and establish a common means of measuring effort, the Doty model, as programmed into INCO's own cost model (described in section 2.3), was used to cross-check the manager's measures. Hence, the model was used to estimate effort measures for both the IWTS as well as the NMIC. Surprisingly enough, the model's estimate for the IWTS was in very close agreement with the actual figures. This result increased confidence in the use and applicability of the Doty model to this particular analysis.

The Doty cost model estimated the IWTS development cost to be on the order of $225,000 over ten months; in contrast, the IWTS management supplied a figure of $200,000 over one year. The environmental factors considered for this Doty run are listed in Figure 3-34. The NMIC estimate came out to be roughly $10 million over 18 months and the factors applicable to this estimate appear in Figure 3-35. In order to estimate the cost benefits resulting from the use of the IWTS system as a scaled prototype, the NMIC data was input to the cost model again, with

```
From the Doty & Associates (RADC) Studies:

                Please Select an Application Category:

                    1  -  Utility (OS)
                    2  -  Command & Control (c2)
                    3  -  Scientific
                    4  -  Business
                    5  -  All (Others not listed above)
                    6  -  EXIT (Return to master CM menu)

                Selection (1-5)? 2
                Estimated Deliverable Source LOC (1,000's)? 15

    Please input a yes/no (Y/N) response to each of these 14 questions:


    Special display? Y
    Detailed definition of operational req'mts? Y
    Changing operational req'mts? N
    Real time operation? N
    CPU memory constraint? Y
    CPU time constraint? N
    First S/W developed on CPU? N
    Concurrent development of ADP H/W? N
    Interactive development environment? Y
    Off-site development computer facilities? Y
    On-site development computer facilities? N
    Development computer different than target computer? Y
    Multi-site development computer facilities? N
    Unlimited programmer access to computer facilities? Y

  56.0 Person Months req'd for analysis, design, code, debug, test and checkout.
            (  Standard error on this approximation = 41.1 %  )

                    Estimated schedule duration =   9.9  Months

                                Continue (Y or N)?
```

Figure 3-34. IWTS Cost Estimate

```
From the Doty & Associates (RADC) Studies:

                Please Select an Application Category:

                    1  -  Utility (OS)
                    2  -  Command & Control (c2)
                    3  -  Scientific
                    4  -  Business
                    5  -  All (Others not listed above)
                    6  -  EXIT (Return to master CM menu)

                Selection (1-5)? 2
                Estimated Deliverable Source LOC (1,000's)? 80

    Please input a yes/no (Y/N) response to each of these 14 questions:

     Special display? Y
     Detailed definition of operational req'mts? N
     Changing operational req'mts? Y
     Real time operation? Y
     CPU memory constraint? Y
     CPU time constraint? Y
     First S/W developed on CPU? Y
     Concurrent development of ADP H/W? Y
     Interactive development environment? Y
     Off-site development computer facilities? N
     On-site development computer facilities? Y
     Development computer different than target computer? N
     Multi-site development computer facilities? N
     Unlimited programmer access to computer facilities? N

  2115.1 Person Months req'd for analysis, design, code, debug, test and checkout.
              (  Standard error on this approximation = 41.1 %  )

                    Estimated schedule duration =  18.1  Months

                                Continue (Y or N)?
```

Figure 3-35. AMIC Cost Estimate

the adjustment of three of the environmental factors to account for the positive impacts resulting from the use of the scaled system. The three factors adjusted were: firmness of system operational specifications, absence of changing operational requirements, and the absence of parallel hardware development. The factors input to the Doty Model are illustrated in Figure 3-36. Amazingly enough, the Doty cost model provided an estimate of approximately $3.5 million for the NMIC development resulting from the benefit of a scaled system - an estimated savings of approximately $6.5 million dollars! With such estimated savings, the NMIC could have cost-effectively afforded the equivalent of twenty-six IWTS development efforts. A conservative figure of $250,000 was used as the development effort cost for the IWTS and the resulting scale factor equation was:

$$\frac{\$250K}{\$10\ \text{Million}} \sim 2.5\ \%\ \text{Scale Factor for Development Effort}$$

The resulting scale factors for the IWTS versus the NMIC are graphically illustrated in Figure 3-37.

```
From the Doty & Associates (RADC) Studies:

                Please Select an Application Category:

                    1  -  Utility (OS)
                    2  -  Command & Control (c2)
                    3  -  Scientific
                    4  -  Business
                    5  -  All (Others not listed above)
                    6  -  EXIT (Return to master CM menu)

                Selection (1-5)? 2
                Estimated Deliverable Source LOC (1,000's)? 80

   Please input a yes/no (Y/N) response to each of these 14 questions:

    Special display? Y
    Detailed definition of operational req'mts? Y
    Changing operational req'mts? N
    Real time operation? Y
    CPU memory constraint? Y
    CPU time constraint? Y
    First S/W developed on CPU? Y
    Concurrent development of ADP H/W? N
    Interactive development environment? Y
    Off-site development computer facilities? N
    On-site development computer facilities? Y
    Development computer different than target computer? N
    Multi-site development computer facilities? N
    Unlimited programmer access to computer facilities? N

 783.2 Person Months req'd for analysis, design, code, debug, test and checkout.
            (  Standard error on this approximation = 41.1 %  )

                Estimated schedule duration =  18.1  Months

                Continue (Y or N)?
```

Figure 3-36. NMIC Cost Estimate with Benefit of a Scaled System

-A CASE STUDY

INCO IWTS / NMIC

INCO IWTS - A SCALED VERSION OF THE NMIC TO BE USED FOR TRAINING AND REMOTE STAND-ALONE WORKSTATION PURPOSES.

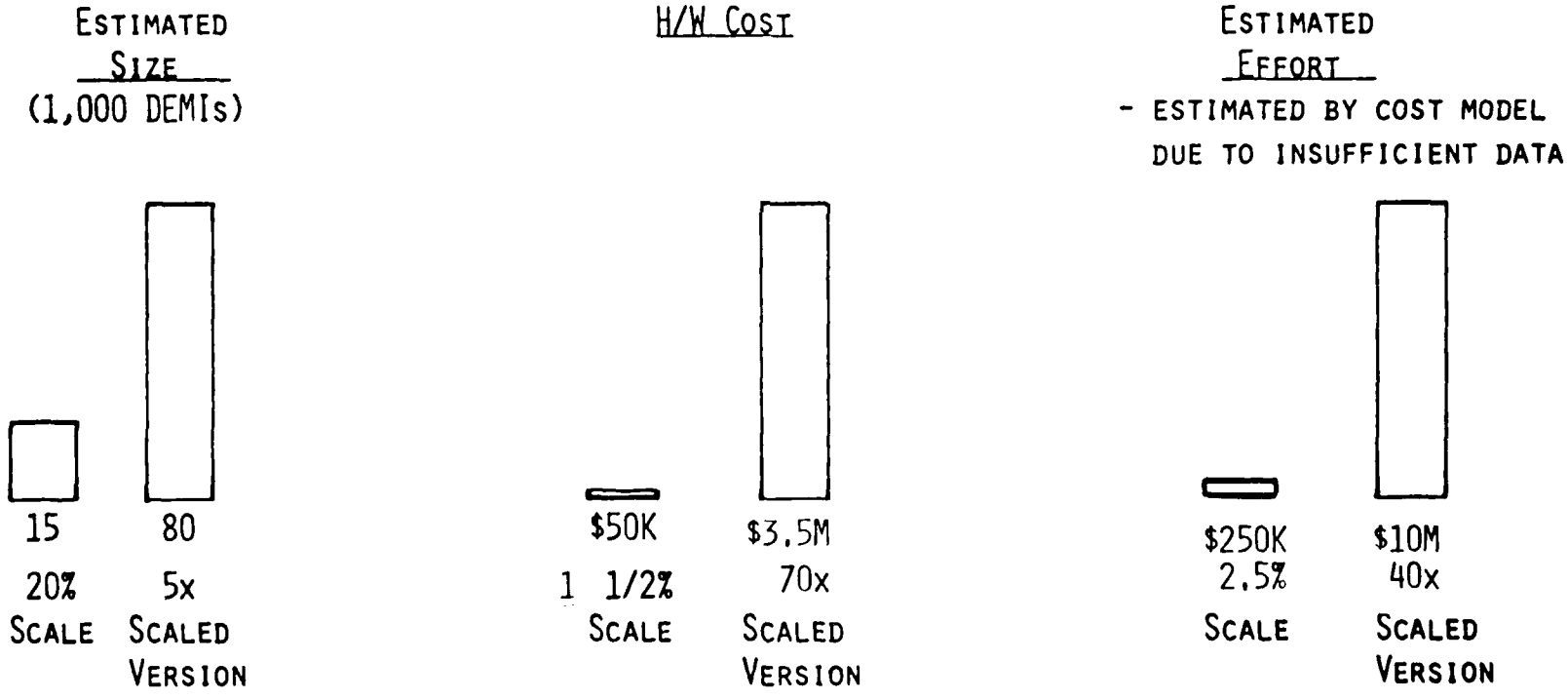


Figure 3-37. Scale Factor Determination

SECTION 4. REMAINING RESEARCH

In this research effort, parameters of software systems that are suitable for scaling have been identified and metrics have been defined for them. These scale factors have then been related to the parameters of the operating system performance simulator. It would be worthwile and advantageous to further develop the simulator to make its parameters more sensitive to the particular requirements of IDHS, i.e., refine its design to be less general purpose and more IDHS-specific. These refinements would aid significantly in analyzing the particular scaled system needs of IDHS.

In addition, a complete set of equations relating system parameters to scale factors would be of great value. The philosophy of this approach and initial delineation of a subset of the simulator variable-scale factor equations are described in "Interrelationships Among Scaling Factors" (Appendix D), and "Simulator Variable-Scale Factor Equations" (Appendix E). Expressing all the input parameters as fairly simple analytic functions of the scaling factors would permit more extensive and definitive analysis of scale factor interrelationships, in order to give the system designer a better tool for evaluating full-system expectations based on those of the scaled system. It would also enable further investigation of how scale factors behave in different operating regions to be performed in a manner which would help in eliminating the uncertainties of the interplay of more than one factor, i.e., the effects of combinations of parameters.

As illustrated in Figure 4-01, the enumeration of guidelines for scaling factors, together with the derived performance relationships, and

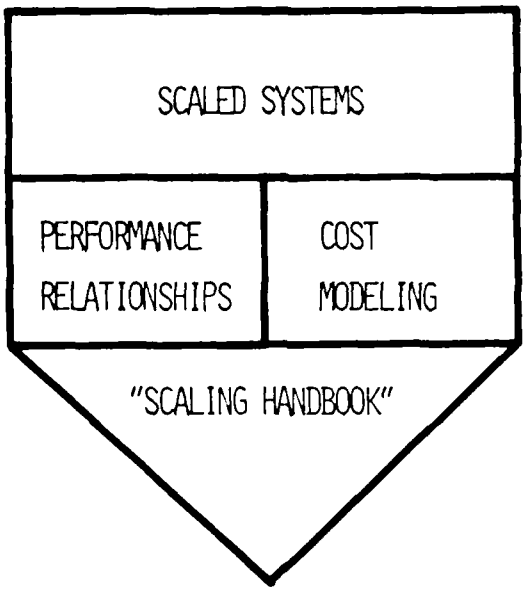


Figure 4-01. The Scaling Handbook

cost modeling results would be used to produce a "scaling handbook", which would be of great value in the design of IDHS. Further work should be done to codify the results of this research to produce the handbook, as well as to verify the efficacy of the scaled systems methodology by experimental means.

APPENDIX A

SCALED SYSTEMS
COST EFFECTIVENESS

15 April 1981

Peter Grimes

Scaled Systems

Cost Effectiveness

August 15, 1980

Prepared by:

Peter Grimes

Advanced Systems
Research Group

INCO, INC
7916 Westpark Drive
McLean, Virginia 22102

This research paper was supported by the United States Air Force contract number F30602-80-C-0219 for the Rome Air Development Center (RADC), Griffiss Air Force Base, Rome, New York 13441.

This document was prepared on an INCO RP-8000 microcomputer system with INCO's Micro-Text-Processor and Micro-Text-Formatter software packages.

Glossary

Full-Scale Effort - See "Scaled Effort".

Scaled Effort - the actual or projected manpower-related effort required to construct a system to a certain scale. If the scale is 100% (or 1:1), then the corresponding effort required to construct the system is referred to as the "full-scale effort"; conversely, if the scale is less than 100% (or 1:n, where n is greater than one), then the effort is "scaled" in the sense that it is, in some measure, less than the corresponding full-scale effort.

Scaled System - an operational system that differs from an ultimate full-scale system in magnitude or degree of functional or operational sophistication and that can be quantitatively related to that system by a scale percentage rate, fraction, or ratio (ie. "50% scaled", "1/2 scale", or "scaled 1:2").

Scaled System Development Effort - the process that results in an operational system built to a relative scale with respect to a target system. The purpose of a scaled system is to serve as a testbed for detecting design deficiencies in the target system so the necessary changes can be made in the front-end of the development cycle, where changes are less costly to effect than in the tail-end of the development cycle.

Scaled System Development Methodology - the formalization of those processes that comprise a scaled system development effort and provide the theoretical foundation for such an effort. The methodology has two principal phases: one to construct and evaluate a scaled system, and another to construct the desired target system.

Un-Scale - See "Up-Scale".

Up-Scale - the process that incorporates evaluative and design factors of a scaled system to the development of the desired full-scale (target) system. The expended or projected manpower necessary to build a full-scale system, given a scaled system, is referred to as the "Up-scaling effort" or "Un-scaling effort".

Abstract

A study has been conducted to assess and quantify the cost impacts of adopting scaled system software development techniques. First, various software cost estimation relationships and models appearing in the open literature were surveyed and evaluated. The results of the evaluation were summarized in tables comparing each model's output (in terms of total developmental effort) given inputs (expressed in deliverable source lines of code (DSLOC)) varying over a range of system sizes. A model (by Doty) was subsequently chosen (due to its consideration of environmental factors) to evaluate cost impacts of scaled system software development efforts versus unscaled, or "full-scaled", development efforts. Preliminary results indicate that substantial cost benefits can be achieved though the use of scaled systems developmental techniques. These results are presented in tables showing costs of various scaled development efforts versus unscaled efforts for projects of varying magnitudes.

From the study, a need was identified to further investigate the subject after the development of cost estimation models more specifically suited to the evaluation of environmental and productivity impacts arising through the use of scaled system software development methodologies. There is also a need to develop quantitative models to account for the benefits of scaling additional aspects of a software system, such as complexity, reliability, and data base.

Background

This study is an outgrowth of current research investigating software cost estimation methodologies and scaled systems software development benefits. Cost estimation is an integral factor in the research of scaled

systems because cost is a principal concern (along with quality assurance and schedule/risk minimization) in the contemplation of scaled systems development techniques.

Scaled system methodology partly consists of implementing and delivering an operational, or semi-operational, software system "to scale". Such a system probably would not support all of the operational and structural characteristics of the desired system but would serve as a skeletal testbed for the system's engineers and customer personnel to evaluate the functional, operational, and performance characteristics targeted for the ultimate product -- a "full-scale" system. Conceptually, scaled systems are similar to the scaled prototype models used by product designers and engineers in the shipping, aircraft, and automobile industries involved in the development of competitive, reliable, and quality products.

The potential benefits of applying scaled system technology to a software development project are many. First, customer functional and operational requirements may be refined and/or solidified through the benefit of a "hands-on" evaluation of the scaled prototype system. Second, in pursuit of performance or reliability increases or to verify that full-scale system performance will be within specifications, the target system design may be optimized. Evaluation of the scaled prototype may reveal potential efficiencies or economies in the system's architectural structure and its real-time management of on-line resources. Third, uncertainty about system technological feasibility, architectural soundness, or reliability may be reduced or eliminated through the experience provided by the scaled effort. Rather than being a costly (and wasted) by-product, experience gained in the scaled effort is

economically realized through the increased productivity of system designers and programmers in the effort expended to construct the ultimate target system -- the "unscaling" effort. In this way, experience is capitalized upon through feed-back. Additionally, experience aids project managers in their decision process by reducing the uncertainty concerning the formulation of the schedule for the unscaling effort.

To some degree, the attractiveness of scaled systems implementation techniques is intuitive. The thrust of this study is to justify the scaled sytems approach based upon quantitative prediction of cost or schedule savings.

Discussion

At the onset, this research started with a survey of current literature in search of software engineering predictive models. A complete list of titles is included in the bibliography. Major titles included "Workshop on Quantitative Software Models" published by the IEEE, "Quantitative Software Models" by DACS, and "Elements of Software Science" by Maurice Halstead. The models of Walston and Felix (IBM), Putnam, Doty, and the System Development Corporation were selected to be encoded into BASIC for execution on a RP-8000 microcomputer system. These were selected primarily for their simplicity; they are all regression-derived equations of the form: effort = constant times number of instructions raised to an exponent (except for an alternate form offered by Doty which computes an additional adjustment factor based upon fourteen environmental characteristics). Additionally, two theoretical approaches derived by Maurice Halstead to quantitatively estimate program effort were implemented. The first appears in chapter eight of his book under the subtitle of "Timing Equation Approximations" [3],

while the other appears in a reader's response to Walston and Felix's article "A Method of Programming Measurement and Estimation" [9], submitted by Professor Halstead.

One objective of scaled system technology is to reduce the uncertainty of software development. This uncertainty is often accounted for in one way or another by current models through "environmental" factors or constraints. Examples of these factors include Doty's attention to the existence of a detailed operational definition or the presence of changing customer requirements and Price-S's \1 "Complexity" input parameter.

Another objective of scaled system technology is to take advantage of increased programmer productivity during the unscaling stage of the software development cycle. Programmer inexperience has been shown to represent a considerable burden on a software development effort over the entire development schedule. This burden may be reduced or eliminated during the unscaling effort as a result of programmer experience gained through the scaled effort, yielding considerable positive cost and schedule impacts.

At this point, we can begin to formulate an approach to estimating scaled system savings. This approach examines the sensitivity of cost (effort) to environmental and productivity considerations.

Approach

As mentioned, six models for programming effort were selected for the study; four derived from regression analysis and two from an interpretation of the natural laws governing human preparation of computer programs (Halstead). Certainly, other models exist and more are currently under development; these were, however, the most accessible for quick implementation. Although accuracy was not a primary consideration

1 Price-S is a proprietary parametric software development cost modeling package invented by, and available for lease from, the RCA Corporation of Cherry Hill, New Jersey.

(the results were not explicitly intended to be used for a cost proposal), representation of the basic relationships existing between program size and effort over a range of program sizes was seen to be crucial to a trade-off analysis of scaled systems technology; therefore, a comparison study was in order.

Various program sizes ranging from one thousand to one million source lines of code were systematically selected and input into the cost models and their corresponding outputs were recorded. The results appear in Table 1. The last column records the mean values of the estimates, their standard deviations, and a disparity factor which was computed by dividing the standard deviations by the means. Because the Halstead model appeared to be quite unstable over such a wide range of system sizes, its results were not included in the correlation computations; only the four regression models' outputs were used in these computations. Disparity factors for the various model's outputs ranged from 4% to 46%, varying directly with program size. For program sizes less than 100k source lines, the disparity factor did not exceed 30%. These figures show that, for systems smaller than 100k, the models are in relatively close agreement.

For this comparison study, the models were to be run without regard to application. This assumption particularly impacted the Doty model, which offers the option of selecting one of several different application categories. For a description of the Doty model's quantitative parametrics, see Exhibit 1. The figures for the Doty model in Table 1 result from the selection of the "All" application category. Table 1a is provided as additional information. It shows the results when the Doty model is run with the selection of the "Command and Control" application category, the proper application selection for software

projects in the DoD environment. Note, however, that the disparity factors are greater in Table 1a due to the use of this application category in the Doty model.

Through the facility of straightforward input-output models such as those described which yield estimated effort given projected system size, only a limited approach to analyze scaled system trade-offs may be formulated. Such an approach is summarized in Table 2, using the Doty model in the "All" application category (Table 2a shows the results of the "Command and Control" application category). Given an estimated full-scale system and its associated predicted effort measure, the efforts required for implementing the system scaled from 10% to 90% were computed. The next step was to estimate the effort for the unscaling effort. In the absence of environmental and productivity computational factors, the estimated unscaling effort would equal the full-scale estimated effort and, hence, no cost savings would be reflected in the analysis. Certain savings factors would therefore have to be assumed. The method chosen for Table 2 was to reflect unscaling effort savings through a reduction in projected deliverable source lines of code. This assumption appears to be a valid representation of the anticipated increased programmer productivity occurring during the unscaling effort. Reducing the number of instructions shortens schedule much as if programmer productivity were increased. If sought-after technical design economies are achieved, they too could be reflected by a reduction in delivered source code for the unscaling effort. Additionally, with the existence of an iterative enhancement development approach [7], some of the design and code for the unscaling effort would be completed prior to the start of the unscaling effort again resulting in effort and schedule savings.

Assuming the validity of these assumptions, Table 2 shows matrices cross-listing scaled effort with net unscaling savings expressed as a reduction in total deliverable source statements. As the table shows, scaled system savings for any system size result when unscaled effort savings equal or exceed the scaled system factor used during the scaled effort. For example, given a scaled system of factor 30 (30% of the total anticipated system size), the table predicts that total project savings will result if the system is scaled and at least 30% savings can be realized during the unscaling effort (this is only slightly different in the "Command and Control" table, Table 2a).

Fortunately, and for the sake of better estimates of scaled system savings, models which incorporate environmental and productivity factors into their computations are available. One such model is a variation supplied by Doty Associates. As with the other Doty models, this model offers five application categories: utility, command and control, scientific, business, and "all". In addition, fourteen environmental factors are accounted for in the computation of projected total effort. These fourteen factors are listed in the literature (reproduced in Exhibit 2) and also in the Doty model's screen display shown in Exhibit 3. Of these fourteen factors, two were deemed most relevant to an analysis of scaled systems savings; these are termed "detailed defintion of operational requirements" and "changing operational requirements".

In the development of modern software systems, more often than not a detailed definition of the operational requirements is lacking; therefore, the environment is one of changing operational requirements. This phenomenon has been attributed to many reasons, but the difficulties customer and systems personnel encounter when

attempting to communicate a system operational specification are probably paramount. One objective of the scaled system development approach is to aid these personnel in arriving at the system's operational specification, and to allow them to economically modify or enhance it, through the benefit of a scaled prototype system which they may evaluate.

Tables 3 and 3a illlustrate the impact of these environmental factors upon total estimated effort, given a discrete system size. The number in the column labeled "Estimated Full-Scale Effort" gives the estimated development effort in the absence of a detailed operational specification and in the environment of changing requirements (see Exhibit 3 for the environmental responses input to the model to arrive at these figures). For the figure appearing in the column entitled "Unscaling Effort", these constraints were removed (Exhibit 3a lists the responses used to characterize the unscaling environment). This figure represents the additional effort necessary to construct the full-scale system, having completed the scaled system implementation and evaluation. The difference in the two figures is the amount of effort which is economically available for the scaled development effort. Estimated efforts based upon the various scale factors are also listed. In arriving at these figures for the scaled efforts, the constraints of no detailed operational specifications and changing requirements were included in the analysis (see Exhibit 3). Another constant relationship was found regardless of system size -- a scaled effort of factor 10 (40 for the "Command and Control" application category) was necessary for any significant effort savings to be realized. The saved effort, however, was significant. For example, in the 100k case, the model showed that savings of

approximately 18 person months could be realized if a 1/10 scaled system could be implemented and a detailed operational specification developed as a result. Assuming a cost of $5000 per person month, this savings translates to a total of nearly $90,000. Exhibit 4 graphically portrays the relationships expressed in Table 3. Of significance is the fact that the data of Table 3 assumes no productivity changes between the full-scale and scaled approaches, which, if present, could even more dramatically increase developmental savings. The "Command and Control" application category of the Doty model predicted a higher break-even scale factor point than the "All" category (see Exhibit 4a). This can be attributed to the greater exponent found in the effort algorithm and the different weights offered for the environmental factors. Exibit 5 offers a generalized portrayal of a scaled system break-even cost/benefit analysis.

Conclusions

The limited study done here with the aid of simplistic cost estimation models alludes to significant cost savings resulting from the use of scaled system development methodologies. None of the analytical approaches presented here, however, account for beneficial productivity changes anticipated for unscaling efforts. From Exhibit 6, the data of Walston and Felix of IBM project 50-180% productivity increases based upon programmer experience. Future work in the areas of software engineering and cost modeling with attention to cost and schedule drivers and productivity factors will benefit system architects both in schedule estimation and scaled system methodology analysis. Research with more accepted parametric models such as Price-S might lead to greater insight into the potential cost benefits derived from the use of scaled system technology. Attention should be

paid, however, to possible parametric impacts of such factors as complexity, reliability, and data base which may require the development and use of additional parametric relationships.

Bibliography

Texts

1. DACS, Quantitative Software Models, SRR-1, IIT Research Institute, Rome, NY, 1979.

2. Gilb, Tom, Software Metrics, Winthrop Publishers, Cambridge, Mass., 1976.

3. Halstead, Maurice H., Elements of Software Science, Elsevier North-Holland, New York, NY, 1977.

4. IEEE, Workshop on Quantitative Software Models, IEEE Publishing Co., New York, NY, 1979.

5. Phister, Montgomery Jr., Data Processing Technology and Economics, 2nd Edition, Santa Monica Publishing Co., Santa Monica, CA, 1977.

Technical Reports / Research Papers

6. J. W. Bailey and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures", Fifth International Conference on Engineering, San Diego, CA, 1980.

7. V. R. Basili and A. J. Turner, "Iterative Enhancement: A Practical Technique for Software Development", IEEE Transactions on Software Engineering, Vol. SE-1, No. 4, pages 390-396, December 1975.

8. A. Fitzsimmons and T. Love, "A Review and Evaluation of Software Science", ACM Computing Surveys, Vol. 10, No. 1, pages 3-18, March 1978.

9. C. Walston and C. Felix,"A Method of Programming Measurement and Estimation", IBM Systems Journal, Vol. 16, No. 1, 1977.

Exhibit 2 Software Development People Resource Estimating Algorithm Methodology

APPLICATION	PHASES OF DEVELOPMENT: CONCEPT FORMULATION PHASE	PHASES OF DEVELOPMENT: ANALYSIS AND DESIGN PHASE
ALL SOFTWARE		
- OBJECT	$PM = 4.790\ I^{0.991}$	$PM = 4.790\ I^{0.991}$
- SOURCE	$PM = 5.258\ I^{1.047}$	$PM = 5.258\ I^{1.047}$ (for $I \geq 10{,}000$) $PM = 5.258\ I^{1.047}$; (for $I < 10{,}000$)* $PM = 2.060\ I^{1.047} \prod_{j=1}^{j=14} f_j$
COMMAND AND CONTROL		
- OBJECT	$PM = 4.573\ I^{1.228}$	$PM = 4.573\ I^{1.228}$
- SOURCE	$PM = 4.089\ I^{1.263}$	$PM = 4.089\ I^{1.261}$ (for $I \geq 10{,}000$) $PM = 4.089\ I^{1.263}$; (for $I < 10{,}000$)* $PM = 0.501\ I^{1.263} \prod_{j=1}^{j=14} f_j$
SCIENTIFIC		
- OBJECT	$PM = 4.495\ I^{1.068}$	$PM = 4.495\ I^{1.068}$
- SOURCE	$PM = 7.054\ I^{1.019}$	$PM = 7.054\ I^{1.019}$ (for $I \geq 10{,}000$) $PM = 7.054\ I^{1.019}$; (for $I < 10{,}000$)* $PM = 2.011\ I^{1.019} \prod_{j=1}^{j=14} f_j$
BUSINESS		
- OBJECT	$PM = 2.895\ I^{0.784}$	$PM = 2.895\ I^{0.784}$
- SOURCE	$PM = 4.495\ I^{0.781}$	$PM = 4.495\ I^{0.781}$ (for $I \geq 10{,}000$) $PM = 4.495\ I^{0.781}$; (for $I < 10{,}000$)* $PM = 3.742\ I^{0.781} \prod_{j=1}^{j=14} f_j$
UTILITY		
- OBJECT	$PM = 12.019\ I^{0.719}$	$PM = 12.019\ I^{0.719}$
- SOURCE	$PM = 10.078\ I^{0.811}$	$PM = 10.078\ I^{0.811}$ (for $I \geq 10{,}000$) $PM = 10.078\ I^{0.811}$; (for $I < 10{,}000$)* $PM = 1.744\ I^{0.811} \prod_{j=1}^{j=14} f_j$

*Can be used for budgeting for programs of $I > 10{,}000$

I in thousands of lines PM = Person - Months

See Exhibit 2 for definitions of "f" factors.

TABLE 1: LEVELS OF EFFORT

Program Size (1,000's DSLOC)	MODEL: IBM's Walston & Felix	SDC	Putnam	Doty (Application = All)	Halstead Estimator	Modified Halstead Estimator	μ / σ / σ/μ % for 1.2.3.4
1	5.2	5.5	4.9	5.3	1.5	.5	5.2 .22 4%
5	22.5	23.7	23.4	28.8	44.9	7.7	24 2.4 10%
10	42.3	44.3	46.0	60.0	194.3	24.4	48 6.9 14%
50	182.8	189.7	221.2	328.6	5741.3	348.1	230 58 25%
100	343.6	355.0	435.0	683.6	24506.5	1082.4	454 137 30%
250	790.9	812.6	1063.9	1800.7	165,999	4818.0	1,117 409 37%
500	1486.2	1520.2	2092.6	3746.6	703,112	14,804	2,211 918 42%
750	2149.4	2193.0	3108.3	5751.2	1,633,630	28,500	3,300 1,465 44%
1000	2792.6	2844.2	4115.8	7795.1	2,969,540	45,327	4,386 2,037 46%

FIGURES EXPRESSED IN PERSON - MONTHS

TABLE 1A: LEVELS OF EFFORT

Program Size (1,000's DSLOC)	Model						μ / σ / σ/μ % for 1,2,3&4
	IBM's Walston & Felix	SDC	Putnam	Doty (Application = C^2)	Halstead Estimator	Modified Halstead Estimator	
1	5.2	5.5	4.9	4.1	1.5	.5	4.925 .5214 10.5%
5	22.5	23.1	23.4	31.2	44.9	7.7	25.2 3.49 14 %
10	42.3	44.3	46.0	74.9	194.3	24.4	51.9 13.4 26 %
50	182.8	189.7	221.2	572.0	5741.3	348.1	291 162 56 %
100	343.6	355.0	435.0	1372.8	24506.5	10 824	627 432 69%
250	790.9	812.6	1063.9	4367.3	165,999	4818.0	1759 1510 86 %
500	1486.2	1520.2	2092.6	10481.3	703,112	14,804	3895 3810 98 %
750	2149.4	2193.0	3108.3	17491.2	1,633,630	28,500	6235 6510 104 %
1000	2792.6	2844.2	4115.8	25154.6	2,969,540	45,327	8727 9499 109 %

FIGURES EXPRESSED IN PERSON - MONTHS

TABLE 2

"ALL" APPLICATION

System Size (1,000's DSLOC)	Estimated Full Scale Effort (Person Months)	Scaled Effort (person-months)		Total Effort (person-months)								
		Scale Factor	Scaled Effort	Based upon net unscaling savings of:								
				10%	20	30	40	50	60	70	80	90
1	5.26	90%	4.7	9.4	8.85	8.31	7.76	7.23	6.7	6.17	5.66	5.16
		80	4.15	8.85	8.3	7.76	7.21	6.68	6.15	5.62	5.11	4.61
		70	3.61	8.31	7.76	7.22	6.67	6.14	5.61	5.08	4.57	4.07
		60	3.06	7.76	7.21	6.67	6.12	5.59	5.06	4.53	4.02	3.52
		50	2.53	7.23	6.68	6.14	5.59	5.06	4.53	4.0	3.49	2.99
		40	2.0	6.7	6.15	5.61	5.06	4.53	4.0	3.47	2.96	2.46
		30	1.47	6.17	5.62	5.08	4.53	4.0	3.47	2.94	2.43	1.93
		20	.96	5.66	5.11	4.57	4.02	3.49	2.96	2.43	1.92	1.42
		10	.46	5.16	4.61	4.07	3.52	2.99	2.46	1.93	1.42	.92
10	59.95	90%	53.64	107.2	101	94.8	88.6	82.5	76.4	70.4	64.6	58.9
		80	47.36	101	94.7	88.5	82.3	76.2	70.1	64.2	58.3	52.6
		70	41.12	94.8	88.5	82.2	76.1	70.0	63.9	57.9	52.1	46.4
		60	34.94	88.6	82.3	76.1	69.9	63.8	57.7	51.7	45.9	40.2
		50	28.82	82.5	76.2	70.0	63.8	57.6	51.6	45.6	39.8	34.1
		40	22.76	76.4	70.1	63.9	57.7	51.6	45.5	39.6	33.7	28.0
		30	16.80	70.4	64.2	57.9	51.7	45.6	39.6	33.6	27.7	22.1
		20	10.94	64.6	58.3	52.1	45.9	39.8	33.7	27.7	21.9	16.2
		10	5.26	58.9	52.6	46.4	40.2	34.1	28.0	22.1	16.2	10.5
100	683.63	90%	611.58	1223							736.3	671.5
		80	539.99							731.5	664.7	
		70	468.91						728.5	660.4		
		60	398.41					727.0	658.0			
		50	328.57				727.0	657.1				
		40	259.54			728.5	658.0					
		30	191.49		731.5	660.4						
		20	124.74	736.3	664.7							
		10	59.95	671.5								120
1000	7795	90%	6973.57	13947							8396	7657
		80	6157.25							8341	7580	
		70	5346.74						8306	7530		
		60	4542.83					8289	7502			
		50	3746.55				8289	7493				
		40	2959.36			8306	7502					
		30	2183.42		8341	7530						
		20	1422.36	8396	7580							
		10	683.63	7657								1367

TABLE 2 (CONT'D)

"ALL" APPLICATION

System Size (1,000's DSLOC)	Estimated Full Scale Effort (Person Months)	Scaled Effort (person-months)		Total Effort (person-months)								
		Scale Factor	Scaled Effort	Based upon net rescaling savings of:								
				10%	20	30	40	50	60	70	80	90
64	426.5	90%	381.6	763							459	419
		80	336.9							456	415	
		70	292.6						455	412		
		60	248.6					454	411			
		50	205.0				454	410				
		40	161.9			455	411					
		30	119.5		456	412						
		20	77.8	459	415							
		10	37.4	419								75
256	1846	90%	1651.8	3304							1989	1814
		80	1458.5							1976	1795	
		70	1266.5						1968	1784		
		60	1076.1					1964	1771			
		50	887.5				1964	1775				
		40	701.0			1968	1771					
		30	517.2		1976	1784						
		20	336.9	1989	1795							
		10	161.9	1814								324
512	3842	90%	3436.8	6874							4138	3774
		80	3034.5							4111	3736	
		70	2635.0						4094	3711		
		60	2238.9					4085	3697			
		50	1846.4				4085	3693				
		40	1458.5			4094	3697					
		30	1076.1		4111	3711						
		20	701.0	4138	3736							
		10	336.9	3774								674
768	5897	90%	5275.7	10551							6352	5792
		80	4658.2							6310	5734	
		70	4045.0						6284	5697		
		60	3436.8					6271	5675			
		50	2834.4				6271	5669				
		40	2238.9			6284	5676					
		30	1651.8		6310	5697						
		20	1076.1	6352	5734			-				
		10	517.2	5792								1034

TABLE 2A

"C2" APPLICATION

System Size (1,000's DSLOC)	Estimated Full Scale Effort (Person Months)	Scaled Effort (person-months)		Total Effort (person-months)								
		Scale Factor	Scaled Effort	Based upon net unscaling savings of:								
				10%	20	30	40	50	60	70	80	90
1000	25155	90%	22020	44040						27518	25315	23393
		80	18977						26884	24475	22272	
		70	16032					26513	23939	21530		
		60	13195				26390	23676	21102			
		50	10481			26513	23676	20962				
		40	7907		26884	23939	21102					
		30	5498	27518	24475	21530						
		20	3295	25315	22272							
		10	1373	23393								2746
100	1373	90%	1202	2404						1502	1382	1277
		80	1036						1468	1336	1216	
		70	875					1447	1307	1175		
		60	720				1440	1292	1152			
		50	572			1447	1292	1144				
		40	432		1468	1307	1152					
		30	300	1502	1336	1175						
		20	180	1382	1216							
		10	75	1277								150
10	75	90%	66	132						82	76	70
		80	57						81	73	67	
		70	48					79	72	64		
		60	39				78	70	63			
		50	31			79	70	62				
		40	24		81	72	63					
		30	16	82	73	64						
		20	9.8	76	67							
		10	4.1	70								8.2
1	4.1	90%	3.66	7.32						4.56	4.16	3.88
		80	3.1						4.4	4	3.6	
		70	2.6					4.3	3.9	3.5		
		60	2.1				4.2	3.8	3.4			
		50	1.7			4.3	3.8	3.4				
		40	1.3		4.4	3.9	3.4					
		30	.9	4.56	4	3.5						
		20	.5	4.16	3.6							
		10	.22	3.88								44

TABLE 2A (CONT'D)

"C2" APPLICATION

System Size (1,000's DSLOC)	Estimated Full Scale Effort (Person Months)	Scaled Effort (person-months)		Total Effort (person-months)								
		Scale Factor	Scaled Effort	Based upon net rescaling savings of:								
				10%	20	30	40	50	60	70	80	90
768	18023	90%	15778	31556						19717	18139	16762
		80	13597						19262	17536	15958	
		70	11487					18997	17152	15426		
		60	9494				18988	17004	15159			
		50	7510			18997	17004	15020				
		40	5665		19262	17152	15159					
		30	3939	19717	17536	15426						
		20	2361	18139	15958							
		10	984	16762								1968
512	10800	90%	9454	18988						11855	10909	10093
		80	8148						11543	10509	9563	
		70	6883					11383	10278	9244		
		60	5665				11330	10165	9060			
		50	4500			11383	10165	9000				
		40	3395		11543	10278	9060					
		30	2361	11855	10509	9244						
		20	1415	10909	9563							
		10	589	10083								1178
256	4500	90%	3939	7878						4923	4528	4185
		80	3395						4810	4379	3984	
		70	2868					4743	4283	3852		
		60	2361				4722	4236	3776			
		50	1875			4743	4236	3750				
		40	1415		4810	4283	3776					
		30	984	4923	4379	3852						
		20	589	4528	3984							
		10	246	4185								492
64	781	90%	684	1368						855	786	727
		80	589						835	760	691	
		70	498					815	735	660		
		60	410				820	736	656			
		50	326			815	736	652				
		40	246		835	735	656					
		30	171	855	760	660						
		20	102	796	691							
		10	43	727								86

Exhibit 2 Software Development People Resource Estimating Algorithms Reflecting Development Environment

ESTIMATION FOR PM: / FACTOR		APPLICATIONS									
		ALL		COMMAND & CONTROL		SCIENTIFIC		BUSINESS		UTILITY	
		$2.060\,I^{1.047}\prod_{j=1}^{j=14} f_j$		$0.501\,I^{1.263}\prod_{j=1}^{j=14} f_j$		$2.011\,I^{1.019}\prod_{j=1}^{j=14} f_j$		$3.742\,I^{0.781}\prod_{j=1}^{j=14} f_j$		$1.744\,I^{0.811}\prod_{j=1}^{j=14} f_j$	
		YES	NO	YES	NO	YES	NO	YES	NO	YES	NO
SPECIAL DISPLAY	f_1	1.11	1.00	1.11	1.00	1.11	1.00	1.43	1.00	1.00	1.00
DETAILED DEFINITION OF OPERATIONAL REQUIREMENTS	f_2	1.00	1.11	1.00	1.54	1.00	2.00	1.00	1.00	1.00	1.00
CHANGE TO OPERATIONAL REQUIREMENTS	f_3	1.05	1.00	1.05	1.00	1.05	1.00	1.05	1.00	1.06	1.00
REAL TIME OPERATION	f_4	1.33	1.00	1.33	1.00	1.67	1.00	1.00	1.00	1.43	1.00
CPU MEMORY CONSTRAINT	f_5	1.43	1.00	1.25	1.00	1.25	1.00	1.00	1.00	1.18	1.00
CPU TIME CONSTRAINT	f_6	1.33	1.00	1.51	1.00	1.67	1.00	1.00	1.00	2.32	1.00
FIRST S/W DEVELOPED ON CPU	f_7	1.92	1.00	1.92	1.00	1.92	1.00	1.92	1.00	1.92	1.00
CONCURRENT DEVELOPMENT OF ADP H/W	f_8	1.82	1.00	1.67	1.00	2.27	1.00	1.33	1.00	1.26	1.00
TIME SHARE, YES A YES BATCH PROCESSING IN DEVELOPMENT	f_9	0.83	1.00	0.83	1.00	0.83	1.00	0.83	1.00	0.83	1.00
DEVELOPER USING COMPUTER AT ANOTHER FACILITY	f_{10}	1.43	1.00	1.43	1.00	1.43	1.00	1.43	1.00	1.43	1.00
DEVELOPMENT AT OPERATIONAL SITE	f_{11}	1.39	1.00	1.39	1.00	1.39	1.00	1.39	1.00	1.39	1.00
DEVELOPMENT COMPUTER DIFFERENT THAN TARGET COMPUTER	f_{12}	1.25	1.00	2.22	1.00	1.11	1.00	1.00	1.00	1.43	1.00
DEVELOPMENT AT MORE THAN ONE SITE	f_{13}	1.26	1.00	1.25	1.00	1.76	1.00	1.25	1.00	1.21	1.00
PROGRAMMER ACCESS TO COMPUTER	f_{14}	LIM: 1.00 UNLIM: 0.90		1.00 1.00		1.00 0.67		1.00 0.90		1.00 0.67	

I in thousands of lines

EXHIBIT 3

From the Doty & Associates (RADC) Studies:

Please Select an Application Category:

1 - Utility (OS)
2 - Command & Control (c2)
3 - Scientific
4 - Business
5 - All (Others not listed above)

Selection (1-5)?
Estimated Deliverable Source LOC (1,000's)? < from tables >
(S)cale, (U)pscale, or (O)ption? O

Please input a yes/no (Y/N) response to each of these 14 questions:

	Responses:
Special display?	No
Detailed definition of operational req'mts?	{ No
Change to operational req'mts?	Yes }
Real time operation?	No
CPU memory constraint?	No
CPU time constraint?	No
First S/W developed on CPU?	No
Concurrent development of ADP S/W?	No
Time share, vis-a-vis batch processing, in dev'ment?	Yes
Off-site development computer facilities?	No
On-site development computer facilities?	Yes
Development computer different than target computer?	No
Multi-site development computer facilities?	No
Unlimited programmer access to computer facilities?	Yes

9999.99 Man Months req'd for analysis, design, code, debug, test and checkout.
(Standard error on this approximation = 99.9 %)

Estimated schedule duration = 999.99 Months

Continue (Y or N)?

ENVIRONMENTAL RESPONSES FOR FULL-SCALE AND SCALED EFFORTS

EXHIBIT 3A

From the Coty & Associates (PADC) Studies:

Please Select an Application Category:

1 - Utility (OS)
2 - Command & Control (c2)
3 - Scientific
4 - Business
5 - All (Others not listed above)

Selection (1-5)?
Estimated Deliverable Source LOC (1,000's)? < from tables >
(S)cale, (U)pscale, or (O)ption? O

Please input a yes/no (Y/N) response to each of these 14 questions:

	Responses:
Special display?	No
Detailed definition of operational req'mts?	{Yes
Change to operational req'mts?	No}
Real time operation?	No
CPU memory constraint?	No
CPU time constraint?	No
First S/W developed on CPU?	No
Concurrent development of ADP S/W?	No
Time share, vis-a-vis batch processing, in dev'ment?	Yes
Off-site development computer facilities?	No
On-site development computer facilities?	Yes
Development computer different than target computer?	No
Multi-site development computer facilities?	No
Unlimited programmer access to computer facilities?	Yes

9999.99 Man Months req'd for analysis, design, code, debug, test and checkout.
(Standard error on this approximation = 99.9 %)

Estimated schedule duration = 999.99 Months

Continue (Y or N)?

ENVIRONMENTAL RESPONSES FOR UP-SCALING EFFORTS

TABLE 3

System Size (1,000's DSLOC)	Estimated Full Scale Effort (Person-Months)	Scaled Effort (person-months) Scale Factor	Scaled Effort	Unscaling Effort (Person-Months)	Total Effort with scaling (EDI)	Savings x $5000/PM
1	2.77	90%	2.48	2.38	4.86	
		80	2.19		4.57	
		70	1.91		4.29	
		60	1.62		4.00	
		50	1.34		3.72	
		40	1.06		3.44	
		30	.79		3.17	
		20	.51		2.89	
		10	.25		2.48	$1450
10	30.87	90%	27.64	26.48	54.12	
		80	24.43		50.91	
		70	21.25		47.73	
		60	18.08		44.56	
		50	14.94		41.42	
		40	11.83		38.31	
		30	8.75		35.23	
		20	5.72		32.20	
		10	2.77		29.25	$8100
100	343.93	90%	308.01	295.09	603.10	
		80	272.28		567.37	
		70	236.75		531.84	
		60	201.46		496.55	
		50	166.45		461.54	
		40	131.77		426.86	
		30	97.50		392.59	
		20	63.78		358.87	
		10	30.87		325.96	$89,850
1000	3832.41	90%	3432.13	3288.22	6720.35	
		80	3033.94		6322.16	
		70	2638.09		5926.31	
		60	2244.90		5533.12	
		50	1854.79		5143.01	
		40	1468.35		4756.57	
		30	1086.47		4374.69	
		20	710.64		3998.86	
		10	343.93		3632.15	$1,001,300

TABLE 3 (CONT'D)

System Size (1,000's DSLOC)	Estimated Full Scale Effort (Person-Months)	Scaled Effort (person-months) Scale Factor	Scaled Effort	Unscaling Effort (Person-Months)	Total Effort with scaling (PMs)	Savings X $5000 /PM
64	215.55	90%	193.03	184.94	377.97	
		80	170.64		355 58	
		70	148.38		333.32	
		60	126.26		311.20	
		50	104.32		289.26	
		40	82.58		267.52	
		30	61.12		246.06	
		20	39.97		224.91	
		10	19.34		204.28	$56,350
256	920.23	90%	824.12	789.56	1613.68	
		80	728.51		1518.07	
		70	633.46		1423.02	
		60	539.04		1328.60	
		50	445.37		1234.93	
		40	352.58		1142.14	
		30	260.88		1050.44	
		20	170.64		960.20	
		10	82.58		872.14	$240,450
512	1901.42	90%	1702.82	1631.42	3334.24	
		80	1505.27		3136.69	
		70	1308.87		2940.29	
		60	1113.79		2745.21	
		50	920.24		2551.66	
		40	728.51		2359.93	
		30	539.04		2170.46	
		20	352.58		1984.00	
		10	170.64		1802.06	$496,800
768	2907.0	90%	2603.38	2494.21	5097.59	
		80	2301.34		4795.55	
		70	2001.07		4495.28	
		60	1702.82		4197.03	
		50	1406.91		3901.12	
		40	1113.79		3608.00	
		30	824.12		3318.33	
		20	539.04		3033.25	
		10	260.88		2755.09	$759,550

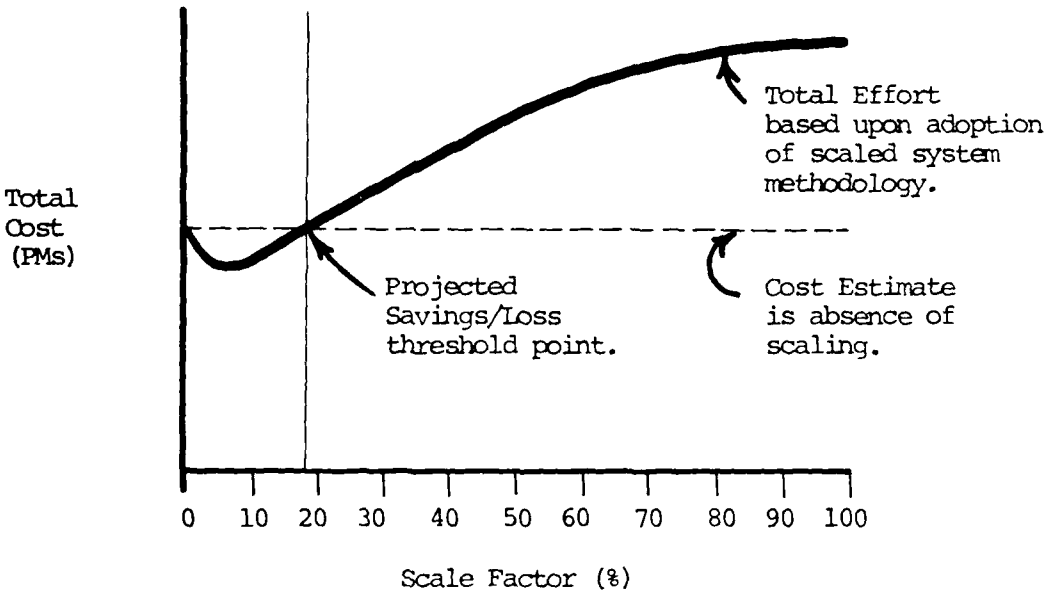
TABLE 3A

System Size (1,000's OSLOC)	Estimated Full Scale Effort (Person-Months)	Scaled Effort (Person-Months) Scale Factor	Scaled Effort	Unscaling Effort (Person-Months)	Total Effort with scaling (PM's)	C^2
1000	5749.64	90%	5033.25	3555.75	8589.00	
		80	4337.54		7893.29	
		70	3664.37		7220.12	
		60	3016.10		6571.85	
		50	2395.74		5951.49	
		40	1807.35		5363.10	
		30	1256.74		4812.49	
		20	753.08		4308.83	
		10	313.79		3869.54	
100	313.79	90%	274.69	194.06	468.75	
		80	236.73		430.79	
		70	200.00		394.06	
		60	164.61		358.67	
		50	130.75		324.81	
		40	98.64		292.70	
		30	68.59		262.65	
		20	41.10		235.16	
		10	17.13		211.19	
10	17.13	90%	15.00	10.59	25.59	
		80	12.92		23.51	
		70	10.91		21.50	
		60	8.98		19.57	
		50	7.14		17.73	
		40	5.38		15.97	
		30	3.74		14.33	
		20	2.24		12.83	
		10	.94		11.53	
1	.94	90%	.82	.58	1.40	
		80	.71		1.29	
		70	.60		1.18	
		60	.49		1.07	
		50	.39		.97	
		40	.29		.87	
		30	.20		.78	
		20	.12		.70	
		10	.05		.63	

TABLE 3A (CONT'D)

System Size (1,000's DSLOC)	Estimated Full Scale Effort (Person-Months)	Scaled Effort (Person-Months) Scale Factor	Scaled Effort	Unscaling Effort (Person-Months)	Total Effort with scaling (P.M.'s)	C^2
768	4120	90%	3606	2548	6154	
		80	3108		5656	
		70	2625		5173	
		60	2161		4709	
		50	1717		4265	
		40	1295		3843	
		30	900		3448	
		20	540		3088	
		10	225		2776	
512	2469	90%	2161	1527	3688	
		80	1862		3389	
		70	1573		3100	
		60	1295		2822	
		50	1029		2556	
		40	776		2303	
		30	540		2067	
		20	323		1850	
		10	135		1662	
256	1029	90%	900	636	1536	
		80	776		1412	
		70	656		1292	
		60	540		1176	
		50	429		1065	
		40	323		959	
		30	225		861	
		20	135		771	
		10	56		692	
64	179	90%	156	110	266	
		80	135		245	
		70	114		224	
		60	94		204	
		50	74		184	
		40	56		166	
		30	39		149	
		20	23		133	
		10	9		119	

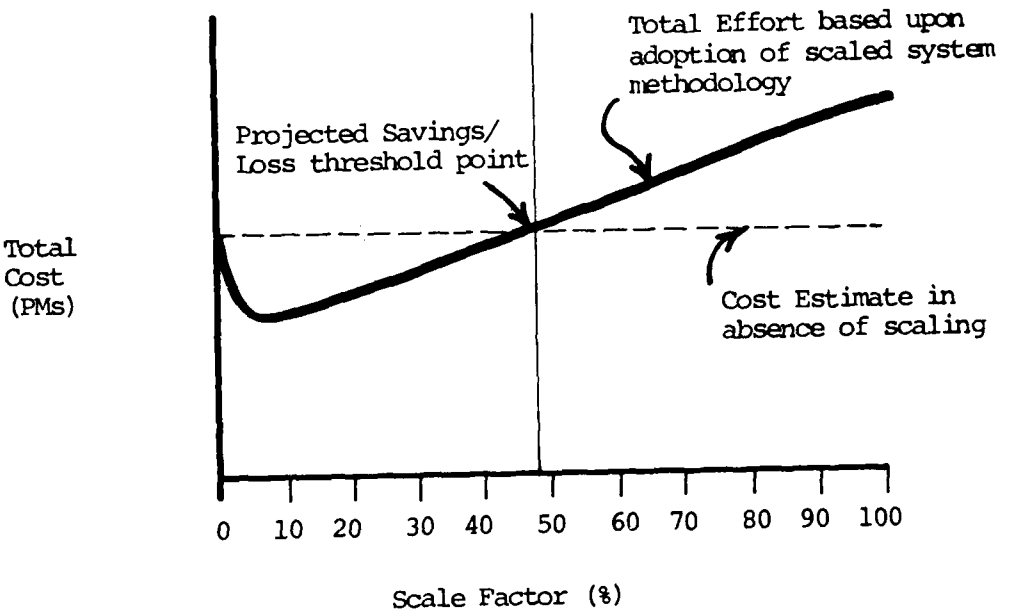
Exhibit 4 "ALL"



- Estimates with considerations of quality of functional requirements definition and environment of changing operational requirements reveal significant cost savings can result from the use of scaled system development techniques.

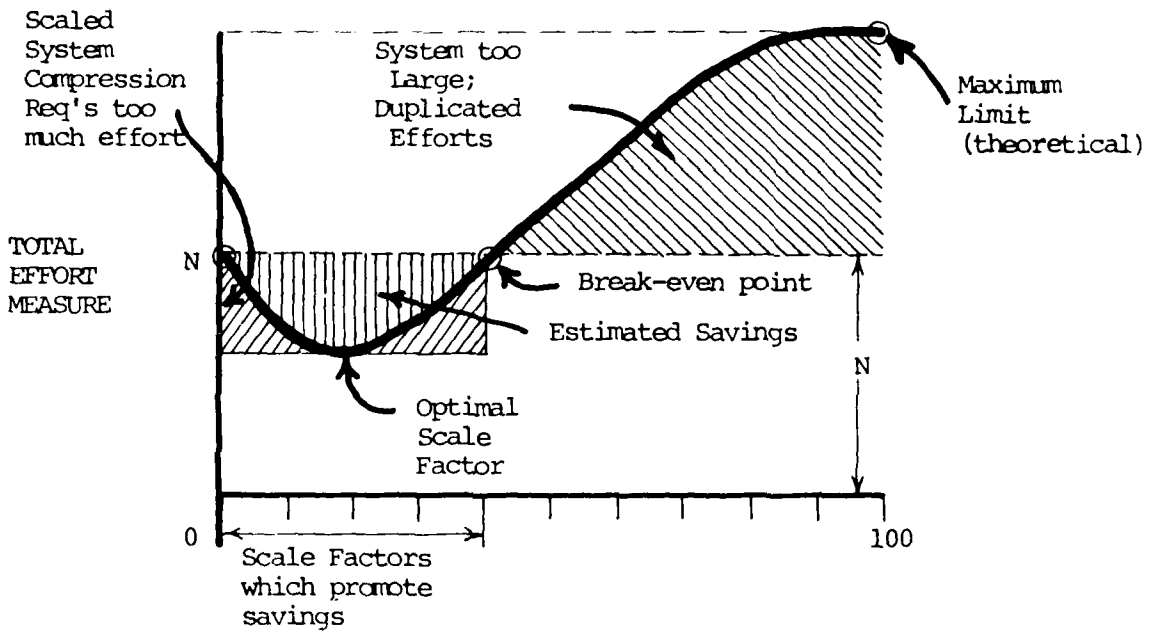
Exhibit 4a

C^2



Estimates with considerations of quality of functional requirements definition and environment of changing operational requirements reveal significant cost savings can result from the use of scaled system developement techniques.

Exhibit 5



Notes:

1. "N" represents the expected effort to design and implement the full-scale version of a proposed system.

2. The curve represents the total cost of a scaled system effort; that is, the cost of the scaled system effort plus the cost of the up-scaling system effort.

EXHIBIT 6

Impact of Experience on

Programmer Productivity

source: Walston & Felix
(IBM)

Without Experience	With Experience	% Increase
146 DSL/PM	Some - 221 DSL/PM	+ 51%
	Extensive - 410 DSL/PM	+180%

DSL = Delivered Source Lines
PM = Person - Months

APPENDIX B

SOFTWARE SCALE PARAMETERS

1 September 1980

Angel Bailey
Thomas Onasch

SOFTWARE SCALE PARAMETERS

1 September 1980

Prepared by:

Angel Bailey
Thomas Onasch

INCO, INC.
8260 Greensboro Drive
McLean, Virginia 22102

This report was prepared in support of contract F30602-80-C-0219 for the Rome Air Development Center (RADC), Griffiss AFB, Rome, New York 13441.

A. ABSTRACT

This paper summarizes the research leading to and involved in the development of a list of software scale parameters. Software scale parameters are those aspects of automated systems that can be reduced in scope in order to implement a cost-effective system scaled with respect to the full-scale system objective.

The work of Yourdon, Tausworthe, Dijkstra, Knuth, Parnas, Mills, Belady, Lehman, Basili, Tinker, Preiser, Halstead, House, Musa, Turner, and others have been studied in an attempt to isolate various elements of software systems most suitable to scaling. Each scale parameter is defined in detail, with sufficient background to introduce the area. Areas for consideration include functionality, data base characteristics, maintainability, security, reliability, performance, language and configuration. These areas will form the foundation for later work under the small scale system design effort.

B. OBJECTIVE

The objective of this report is to identify parameters of software systems subject to scaling and to begin a definition of scale factors associated with each. These scale factors will be used to develop appropriate metrics to standardize, quantify, and objectively describe a scaled system in terms of the full-scaled system it represents.

C. DISCUSSION

An analysis of current software systems development methodologies has been conducted to isolate the elements most suitable to scaling. Considerable research has been performed to identify other work in the software engineering field that would be applicable to the scaled systems

project.

It is particularly important to identify the point in the system development cycle at which it is appropriate to employ a scaled system. A major constraint to this decision is the availability of sufficient information on which to base scale factor decisions. A further limitation is imposed by the need to define requirements to the total system level before scaling to a whole system reference is possible.

Current software system development methodologies emphasize a process in which development is conceived as proceeding through a series of phases. Each phase is organized to complete a specific planned process and produces output in terms of information or design documents which, in turn, is input to the next phase. Referring to the DoD lifecycle description, this process begins with the initiation phase and progresses through the development, evaluation and operation phases. Most attempts to improve the efficiency of the development cycle have concentrated on improving the processes which comprise some single phase. Thus structured programming focuses on the programming stage of the development phase while composite design applies to the design stage of the development phase. The scaled system approach, as it is envisioned, bridges the gap between the definition and design stages of the development phase.

To further clarify this conclusion, consider the activities that make up the definition stage. Robert Tausworthe in Standardized Development of Computer Software calls this the program definition or functional specification phase, which he divides into two activities; that of creating the software requirement (Figure 1) and the software

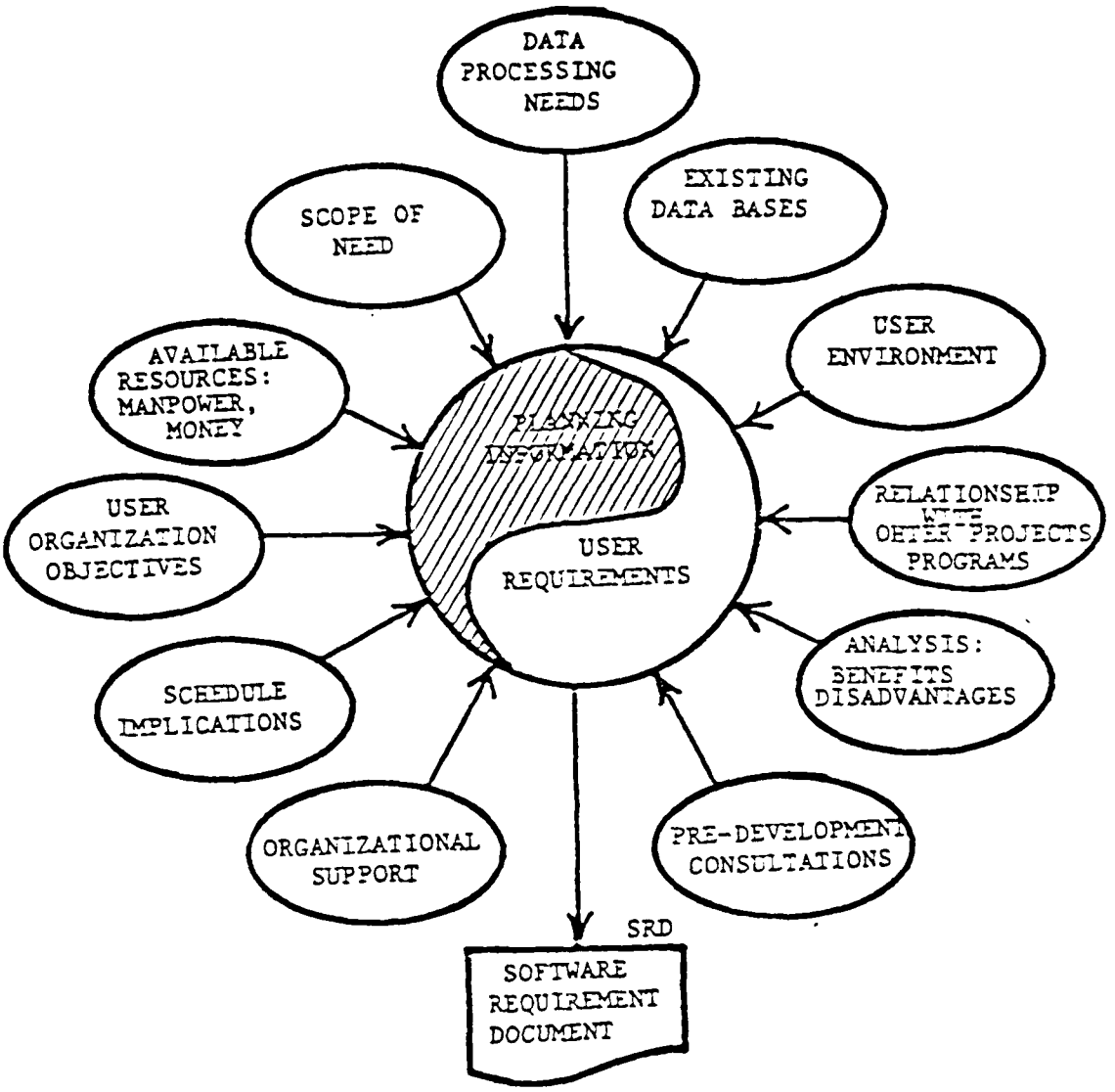


Figure 1. The Software Requirement

definition (Figure 2). As Tausworthe explains it, the creation of the software requirement further consists of two parts, both largely non-technical, to conceptually lay out the requirement. The first part, that of planning information, establishes the requirement for the software. The second part, the user requirements, establishes the requirements of the software.

Following the conceptual activity of software requirement creation comes the functional definition of the software. This is a technical activity which, when complete, defines both what the software is to do (not how it is to do it) and the meaning of program correctness.

The requirements and definition activities are an iterative cycle. Concurrent interaction between requirements, definition and approved amendments is a necessary activity to achieve a final balance betwen software requirements and feasible system definition before the detailed design process begins. It is at this stage that the definition criteria may be applied to the development of scale factors and the preliminary requirements to scaling established. Attempts to define scale factors earlier in the process will suffer from insufficient data. Factor definition at a later point will be constrained by the progress in detailed design. It should be reiterated that definition to the total system level is necessary in order to provide the total system baseline to which one must scale.

D. TECHNICAL APPROACH

The technical approach to the process of deriving scaled parameters from various elements of the software definition is addressed in the remainder of this report. The software elements applicable to the

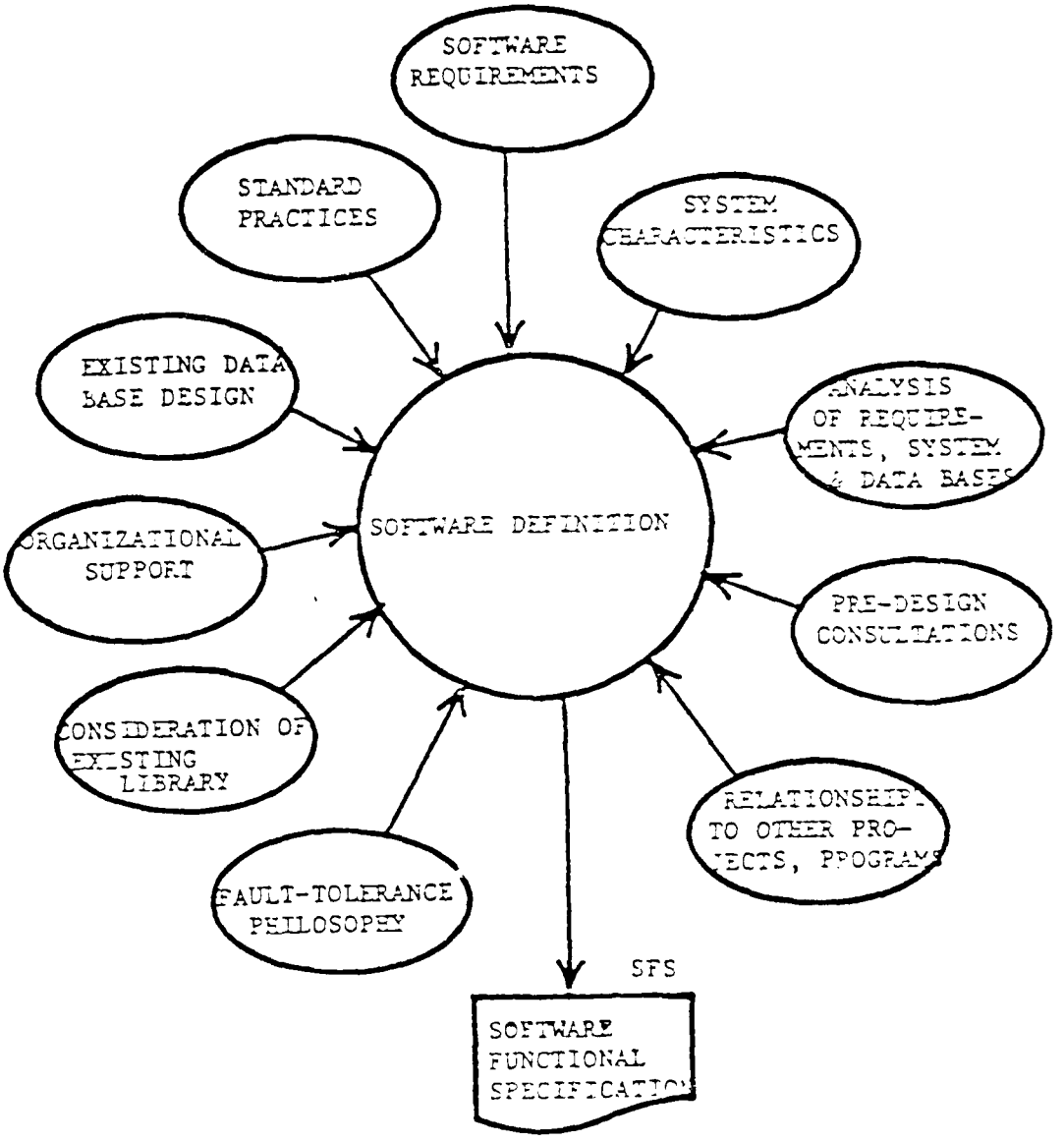
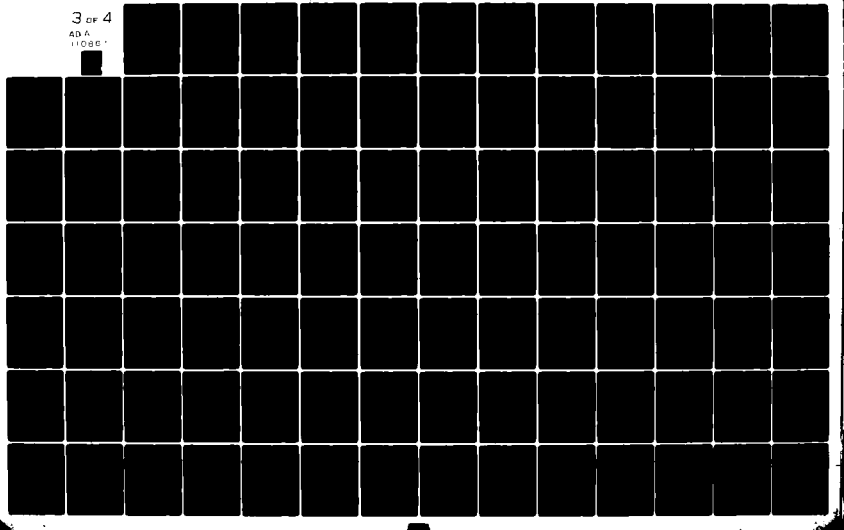


Figure 2. The Software Definition

AD-A110 867 INCO INC MCLEAN VA F/G 9/2
SMALL SCALE SYSTEMS.(U)
SEP 81 M KERCHNER, P GRIMES F30602-80-C-0219
UNCLASSIFIED INCO/1155-681-TR-46-D(F) RADC-TR-81-251 NL

3 OF 4
AD A
110867



scaling of data base, performance, functionality, security, maintainability, reliability, language and configuration are described and defined in detail with examples. It should be noted that there are two components of scaling to be considered. The first is the identification of the scaling parameters themselves such as size, modularity, etc., and the second component is the measured effect on items such as the throughput and utilization of the total system itself.

1. Data Base Characteristics

A data base is a collection of data records between which specific relationships exist. These relationships may be used to link record types and records of the same type. A record is an aggregate of data transcribed, or in a form suitable for transcription, between a computer and an external medium. Each record comprises data (normally called fields) that have an underlying relationship to one another. Data elements in a record may be of similar or dissimilar types; bits, numbers, character strings, etc. Records of the same type are usually grouped into larger aggregates called files. In practice, a large file may contain hundreds or thousands of blocks, each containing one or more records.

Data base scale parameters derive from two areas. The first area concerns the complexity of the access system and the second concerns the various size elements involved at each level of the data base structure.

a. Data Base Complexity

There are at present three major data base access methods that must be examined for scaling purposes. They are, in increasing

order of complexity:

1) Sequential Access

The term sequential access is used when the access to records is through a key by which the file is physically sequenced. Access is therefore serial, i.e., each item must be examined in sequence until a key match is found.

2) Indexed Sequential

The indexed sequential access method (ISAM) refers to a setup whereby an index table is established through which record access is made. In ISAM, one or more items in each record is chosen as the "key". The index then consists of an ordered sequence of the values of the ISAM key which occur in the collection of records that compose the data base. Associated with each value is an address or pointer to the record. The file is stored in some kind of direct access storage such as disk or drum so that once an address is retrieved from the table, the associated record may be accessed directly. Note that although the record may be accessed directly, the process of finding the pointer to the record still involves a sequential (or perhaps binary, if the table is ordered) search of the index table. The advantage of ISAM is that once the address is determined, all data may be accessed with equal ease. ISAM is often used for very long files containing thousands of records. Table search time is considerably less than the time required to search through each record.

3) Random Access

For this access method, no index table is maintained. Instead, to decide where in storage a record may be accessed, the value

of the key is used as input to some hashing algorithm which is designed to produce as output a storage address. This address is not necessarily a physical address in the sense of stipulating exactly which physical location in direct access storage will be used, but may be a logical address within some area.

In the case where the data base is organized in hierarchical form, that is in applications where a natural hierarchy of relationships exist between data items, and any data subset is contained entirely within its superset, another form of access may be used. The root, or parent, record may be located by either a hashing method or by sequential search of a table. Subsequent records "lower" in the hierarchy may then be accessed through direct access pointers. Direct access pointers may also be used in the network model in which the data structure sets serve as the logical links between records of different types and reflect the data organization rather than an exact representation of entities. The data structure may be quite complex in that one record may be linked with any other and have any number of superiors or subordinates.

Another form of direct access occurs with the relational data base. In this model, largely experimental, data are organized into tables (relations) each of which may be directly accessed through the table name. Row and column ordering has no significance and each column (or domain) may be directly accessed.

Data base complexity may be scaled by first employing the simplest access method (sequential) to model a data base and then developing the scaling relationship involved in increasing the complexity

from sequential to indexed sequential to random access. At another level, it is also possible to scale complexity by restricting to single access a system which in full scale would allow multiple access to the data base.

The scaling step from a sequential data base to an indexed sequential is straightforward. Quantitative measurement of results of this scaling step is of course based on size factors as for example, the number of records in the data base. An average access time for a sequential search is directly related to the number of records. For an indexed sequential access only a portion of the access time (the table search) may be directly attributable to the number of records.

The next quantitative step, to a random data base, is expected to be more difficult to scale, as the measurement parameters of a random data base configuration are highly dependent on data base usage and organization requirements. However, given a constancy in data structure, measurement is still possible. Considering a hashing approach to address determination, access time difference between the indexed sequential and random access methods is the difference between the average table search time and the time necessary to execute the hashing algorithm (including time to resolve duplicated references).

Another method for scaling complexity is the depth and complexity of the data structure which describes the relationships between data items. Any number of possibilities exist with this approach. A hierarchical data base could be scaled, regardless of access method, by limiting the number of immediate successors to a node or the branch points at a given level. Alternatively, the number of levels could be

scaled. As yet another example, for a direct access network, the pointer chain could be limited to only the forward direction.

b. Data Base Size

Aspects of data base size are relatively easy to scale, requiring merely numerical quantification of appropriate elements. The various size elements subject to scaling in a data base are:

1) Number of Files

The number of files may be scaled by applying a straight percentage to the total number of files. (Unless all files are of equal length, the total data base size will not necessarily be scaled by this same percentage.)

2) Length of Files (numbers of records)

The number of records may be reduced and the scale factor determined from a ratio of the number of records remaining (in all files) to the total number of records.

3) Number of Access Keys

Scaling could be applied by limiting access to data base through a single prime key. The scale factor in this case is of course, related to the number of secondary keys in the final system.

4) Number of Fields

The number of fields in all records of a given type may be scaled as a ratio of the number of fields remaining to the total number of fields.

5) Length of Record/Field

The length of a record may be scaled by considering a ratio of the number of characters in the scaled record

compared to the total number of characters. The same scaling could be applied to each or selected fields.

It is possible of course, to scale access complexity, data structure complexity and size elements in combination, but measurement of scaling results then becomes increasingly difficult. For example, how would one compare a sequential access, hierarchical, single key data base to a multifile relational data base? As has already been implied, there is mutual dependence between the scaling factors of type and those of size. All of these relationships will be examined at the point in the study where factor quantification is addressed.

A final point on data bases and scaling. Scaling of access type and to some degree, data structure complexity, may be restricted if the system being developed is expected to use an existing data base management system. Even the most sophisticated and general purpose system is restricted in the organization of the data base it can manipulate. Scaling of data base complexity could at some point, involve a data base management system customized to the scaled system.

2. Performance

Performance or efficiency objectives such as response times and throughput rates under a variety of workload and configurations are an important part of most system designs. Efficiency can rarely be specified as an absolute because it is influenced by such factors as the hardware configuration, telecommunication line speeds, the efficiency of all other concurrently executing programs and the number of active terminal users, to name a few.

Performance may be interpreted as the technical equivalent of the economic notion of value. That is, performance is what makes a system valuable to its user. Like value, the concept of performance is a subjective one. This means that different people tend to use different performance indices in assessing systems. However, it is often possible to translate subjective definitions of performance into purely technical terms, which can sometimes be quantified and therefore objectively evaluated.

These elements may be considered to be scaling elements and thus developed and measured for scaled system use; either to be scaled, or to measure the effect of scaling.

The most common classes of quantitative performance indices for computer systems are:

a. Productivity

Productivity is generally defined as the volume of information processed by the system in a unit time. One measure of productivity is the throughput rate, which during a given interval of time, is the average rate at which jobs are completed by the system in that interval.

Throughput may be scaled. If, for example, the full scale system is to process 2000 messages per day, the scaled system might be required to process only 500. This system would be throughput scaled to 25% of the full-scale system.

Throughput is of course, a result of nearly every aspect of a system configuration; from the hardware itself to the functions the system is required to perform to the typical set of jobs requiring system

resources. The system configuration from both a hardware and software viewpoint will be discussed later. In general terms however, consider how throughput might be scaled:

1) System capacity. As the maximum rate which a system can perform work, capacity has a direct result on throughput. The scaled system might handle only jobs with primary memory requirements of 10k as opposed to 100k for the full-scale system; or jobs using less than 30 seconds of processor time versus two minutes; or those using only one printer and a card reader rather than the several I/O devices jobs on the full-scale system might require.

2) System job mix. Although the full-scale system might be required to process some number of job types arriving at random, the scaled system job mix might be structured for optimum performance. Depending on the application, this might mean grouping all jobs of type A together. Conversely, in a multiprocessing environment, since all jobs of the same type might compete for the same resources, types A and B might be alternated in the job stream. As a final example, one could scale by configuring for average expected work load rather than peak load.

b. Responsiveness

The term responsiveness can be defined as the time between the presentation of an input to the system and the appearance of the corresponding output. A measure of responsiveness is the response time, which is the time elapsed between entering a request and the computer's acknowledgement of it. In general, the response time depends on the request, on the system, and on the work load in the system at the time

the request is entered. Nevertheless, response time is a valid parameter to scale. We might require the target system to support 20 analysts with 5 seconds response. For scaled system development, a response time of 15 seconds for 5 users might be adequate. In such a case, the system would be scaled 25% with respect to number of users and 33% with respect to response time.

A better term would be interactive responsiveness (the inverse of response time) or the number of responses per unit time. This keeps a consistency in terminology whereby scaling refers to reducing the value of a parameter. In terms of this example, by scaling interactive responsiveness we are accepting 4 responses per user-minute as compared to 12.

Below a certain threshold on the low end of the scale, human users can no longer appreciate a reduction in response time (an increase in interactive responsiveness). At the other extreme, at some point response times get unacceptably long and the level of user satisfaction drops to the point where longer response times make no difference. Since even on a scaled system, user satisfaction may be of some importance, the degree to which responsiveness is scaled should be limited by the characteristics of the users.

c. Utilization

The term utilization is generally defined as the ratio between the time a specified part of the system is used during a given interval of time. Examples of utilization include hardware module (CPU, memory, I/O channel, and I/O device) utilization, and the utility package utilization.

Modules may be linearly scaled as the ratio between proposed and actual module utilization. The scale factors may be measured in terms normal for the module, e.g., CPU utilization is measured in instructions per time, memory utilization is measured as a percentage of total memory available, I/O channels as either a data rate or channel ratio, etc.

As an example, we might scale utilization by requiring that the developmental system require utilization of only 50% of available capacity. It has been generally shown that this scaling of performance requirements will result in a development cost one-third that of a system requiring 90% utilization of resources (Barry Boehm, Practical Strategies for Developing Large Software Systems). In this case, the system would be utilization-scaled to 55% of that of the target system.

d. Operating System/Organization

Performance scaling may be accomplished on a more fundamental (and probably less quantifiable) level by several other methods. Although the parameters mentioned below represent a mode of operation rather than a measurable ratio and are not always the object of a design effort, they do affect the total system effort. Hence the choice of one mode over another is a valid method to scale performance.

1) Processing mode. Several possibilities come to mind here. Consider a system where batch, interactive and real time requirements must all be supported. Advantages in terms of development time would certainly accrue to a scaled system which considered merely a single mode. Similarly, a real time system such as a tracking network

could be scaled with a batch system which used simulated input data.

2) Operating system. Although the choice of processing modes is certainly dependent on the operating system (or vice versa), the operating system presents other ways to scale. While the ultimate system might require a custom operating system, scaling could be accomplished by choosing an off-the-shelf system or by modifying an existing one. Given that an existing operating system is to be used, one could scale by leaving unnecessary functions in the executive of the scaled system.

3) Interrupt processing. Closely related to other areas such as choice of processing mode, number and type of peripherals, system functions, etc., the mechanisms for interrupt handling may be considered as a separate parameter. Several different approaches are possible. For example, certain (or all) interrupts might be ignored until the CPU is free. Alternately, a priority interrupt scheme could be scaled with a simple system of queued interrupts.

There are a number of other performance affectors such as the ease of use of a system, the structuredness of a program or of a language, and the power of an instruction set. However, they are not considered in the scaling process because they are difficult or impossible to quantify.

3. Functionality

Large programs are often decomposed into a set of interacting functional components (e.g., modules, procedures, subroutines, etc.). This principle by which program concepts evolve in a natural, structured way emerged from Dijkstra's work in the "The Multiprogramming System." He conceived that a program could be organized into hierarchical levels

of support. The principle, known as levels of abstraction, formed the basis for what has since become known as structured programming. At each level of abstraction, it is useful to study the needs of the problem, that is, to identify all the relevant elements of control and data and the relationships between them.

System structure refers to the way in which complex functions and interrelationships may be characterized in terms of successively simpler components sometimes called modules. Structure primarily manifests itself in terms of relationships such as cohesiveness and coupling within and among the systems modules, the architecture of the functions and data flows, and the information structures. Each component forms a natural unit on which to focus attention when attempting to scale the system. Independence of the modules determines the modularity of a system.

Usually, structured software is organized into a master module which calls subordinate modules, which in turn link to modules which are further subordinate and so on down the hierarchical or functional chain. In principle then, the abstract description of a given component will embody infonmation about the entire chain of its subordinate components.

Functional scale factors will depend on the degree of modularity developed in a hierarchical system. The applicability of modular scaling would be dependent upon the type and degree of coupling between modules and levels of modules. There are three types of coupling to consider:

a. Data coupling - a form of coupling caused by an intermodule connection that provides output from one module which serves

as input to another module.

b. Control coupling - a form of coupling in which there is a connection between two modules that communicates control.

c. Hybrid coupling - a strong form of coupling that occurs when one module modifies the procedural contents of another module.

The significance of coupling with respect to scaling is determined by the direction and strength of the connection. If the functional coupling between levels is weak, then the details in the description of the lower level modules rapidly become insignificant with respect to the higher levels. In this case a level in the calling hierarchy may correspond fairly closely to a level of functional description and scaling by modular elimination of a horizontal module chain, (function level) is feasible and is represented by the outline B of Figure 3. This of course corresponds to the elimination of some number of primitive functions across the entire system. The percentage of functions retained could be considered the scale of the system.

In the opposite case, when the functional coupling between levels is strong, scaling by eliminating bottom to top serial structures would be the indicated method. This would be analogous to the elimination of an entire subsystem and is represented by the outline A in Figure 3. Since structured design carries a strong preference for vertical coupling and requires the avoidance of complicated coupling schemes, such as hybrid coupling, functional scaling appears to be a practical method for top-down structured designs. Consider the following examples of functional scaling:

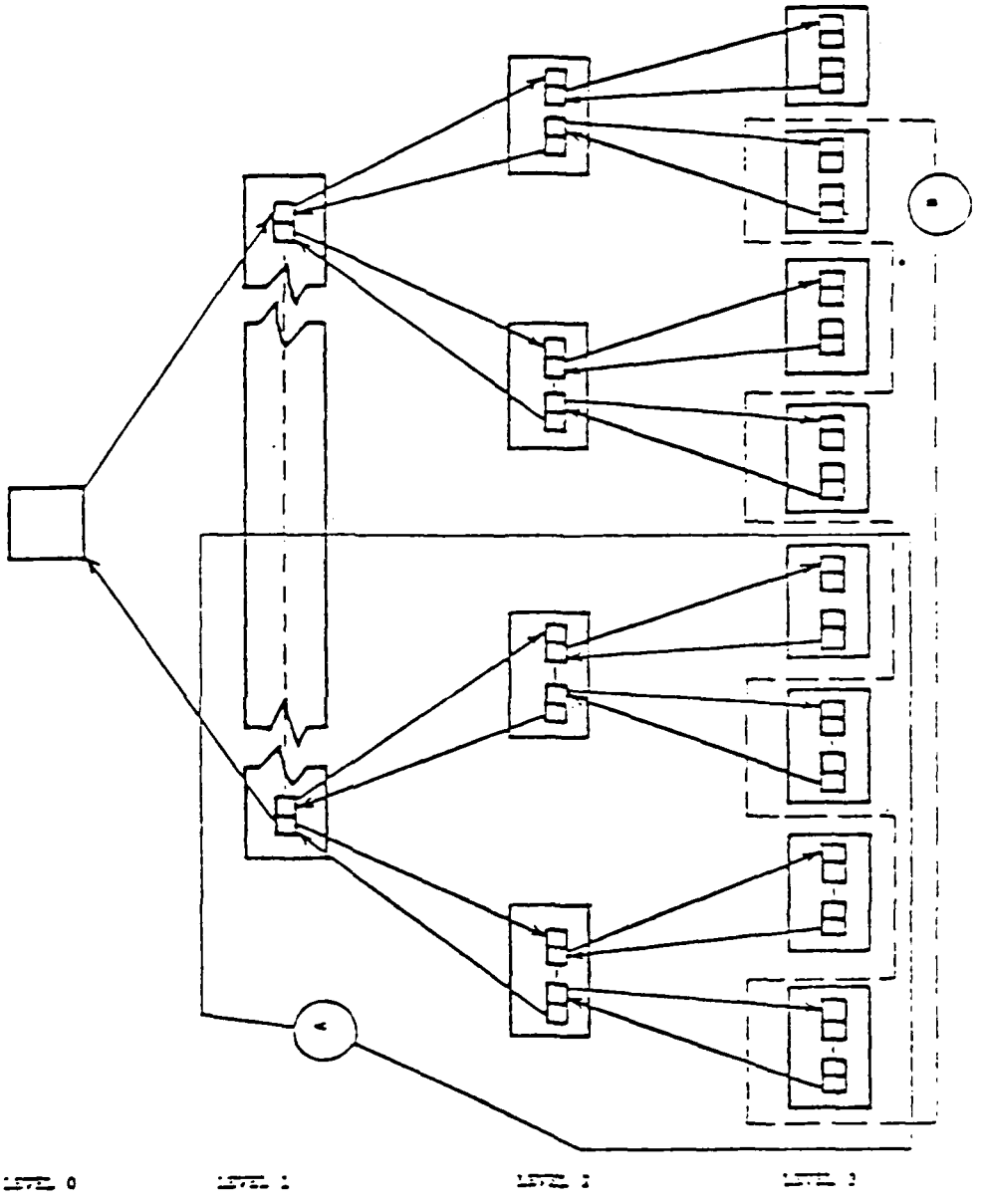


Figure 3

a. Eliminate performance monitors throughout the system.

b. Eliminate utilities which would provide the user with transparency of data (format control, code translation, interfacing, etc.).

c. Eliminate all non-standard OS requirements.

d. Eliminate non-critical ancillary functions.

e. Implement select disjoint subsystems rather than the integrated system.

4. Security

The term security can be defined as the extent to which unauthorized access to software or data by unauthorized persons can be controlled. A user should be able to create and manipulate various types of resources and delegate the access rights to a resource to other users. A legitimate user of a resource is one who has either created it, or obtained permission to use it from another legitimate user. A user should not be able to disrupt the processing of another user in any unauthorized way, as for example, causing him denial of service.

The degree of security provided for software and data is determined by the scope of access control and the completeness of access audit. Access control consists of those attributes of software that restrict access tc and manipulation of programs and data. Access auditing is the procedure whereby an historical record is maintained of both successful and unsuccessful attempts to access restricted data.

Security may be considered a valid parameter for scaling when the scaled system will be developmental in nature and when either adequate physical safeguards may be substituted for the full-scale

software security procedures or the data to be protected is simulated or is non-sensitive public test data.

One approach to scaling security is to modify the file protection procedures implemented to control access. (Methods for identifying the legitimate user will be discussed later.) A system may be considered scaled with respect to security if it encompasses a file protection methodology less restrictive than the full-scale system. The following list (by Randall Jensen in Software Engineering) summarizes six levels of file protection starting with the least sophisticated:

a. No protection - file access and all operations provided any user.

b. Total protection - no file sharing at all.

c. All or nothing - if access granted, then all operations permitted.

d. Controlled sharing - a user is granted access rights which are the minimum necessary to accomplish the specified task.

e. Specified access - access to each object is restricted and access rights are owner-definable in several different contexts:

1) User-dependent - access rights are based on the identity of the user requesting access.

2) Context-dependent - access is granted subject to the environment (type of terminal, location, time of day, etc.).

3) Data-dependent - access to a record is controlled depending on the contents of the record.

f. Post-access control - users may be granted access, subject to the purpose for which it is to be used after access is accomplished.

Aside from the types of operating system-provided protections described above, the "size" of the access specification may be scaled. One method of specifying access is with the access matrix, the dimensions of which (which may be altered) are determined in one direction by the number of users, processes, or procedures which have access restrictions and in the other direction by the number of objects for which access is restricted. Changing either dimension necessarily scales the system with regard to security.

In addition to the file protection scheme provided by all sophisticated executives, one must also consider the classification of the data and the clearances needed by the users. Classified information is commonly protected by a trusted subsystem which evaluates (beyond the OS) the protection afforded and access granted to various classes of sensitive data and programs. Implementation of this subsystem provides several new methods for scaling.

a. To access data, both the user and the terminal must have access rights. To scale, we might grant all terminals access to everything and provide only physical security for access to the terminals. Alternatively we could allow all users free rights to any data.

b. The number of access types (by user, by classification level, or by compartment) could be reduced.

c. Different data sets (further scaled of any method described under data bases) could be provided for each access clearance.

d. The granularity of the data base could be modified for each level of access. In other words, one user class might be granted

full access to all information while another might be restricted to record level. An alternate method would be to restrict access only beyond a certain file structure level across the entire class of users.

e. Codewords and special handling could be eliminated.

f. The audit trail that might be required of the full-scale system could be ignored for the scaled implementation.

g. The authentication approach (to include passwords, recording of access failures, log-on procedures, and terminal authentication) could be simplified or eliminated.

h. Through the use of simulated data or limited transmission, a full-scale requirement for encryption of data (by software) could be suspended.

5. Maintainability

The term maintainability can be defined as the effort required to locate and fix an error in an operational program and is a technically valid area for scaling, since the implementation of maintainability incurs increased software development cost and/or time. The approaches which might be employed to scale this parameter are closely allied to those for scaling functionality in that functional requirements of the system are eliminated or simplified. The difference is that the functions included to enhance the maintainability of a system would not normally be the target of a design effort but rather tools to make subsequent enhancements of the final product a routine process. With this in mind, one could argue that since the scaled system is merely a step toward the final product, modules whose purpose is to enhance maintainability could be excluded from the scaled effort. Some examples

would be:

a. Simplify process-error handling.

b. Eliminate restart/recovery procedures.

c. Eliminate modules to reject and/or correct bad data.

d. Eliminate, reduce or modify fault location/trap software.

e. Exclude software to monitor system performance and gather statistics.

f. Reduce back-up procedures to a minimum.

g. Include additional software diagnostic aids, program tracers, and interactive debuggers. This is an unusual situation in that additions made to the scaled system would reduce the development effort by enhancing its effective implementation. The full-scale system would probably contain built-in diagnostic aids as well, but to a lesser degree.

Thus, the design of the scaled system may eliminate documentation, recovery, and reconfiguration programs at the specific risk that the lack of these elements may in fact prolong the project rather than enhance it. This risk may, in some cases, be sufficient to preclude the scaling of maintenance functions in the scaling process.

6. Reliability

The term reliability refers to the extent to which a program can be expected to perform its intended function with consistency and required precision. Reliability is the product of the error tolerance, simplicity, accuracy and of course, consistency of the software and data.

The scaling of reliability is a tricky business. While it is certainly valid to state that the reliability standards in the scaled

system may be relaxed (and therefore scaled), some of the methods whereby this could be achieved would be poor practice in any design effort, scaled or not. Some of these would be inconsistency in calling sequence and I/O conventions, non-standard data declaration and non-standard design structure. More feasible alternatives would include:

a. Reduction of precision.

b. Elimination of error detection software geared to errors which would occur infrequently in practice or not at all in the input to the scaled system.

c. Use of fast, easy (and not necessarily accurate) approximation functions and algorithms.

d. Relaxation in enforcement of coding standards. This approach might be considered in the case where the scaled design is a skeleton of the final system and recoding would be necessary anyway. When the scaled approach is to implement a complete subsystem, leaving the door open for recoding the subsystem is not a good idea. Whether this approach would indeed, scale reliability is probably a function of the quality of the programmers. Allowing each programmer to "do his own thing" would speed up the development effort but might, if the programmers were good, not significantly affect the reliability of the result. Whether a case can be made for reduced reliability regardless of programmer quality is a question for further research.

7. Programming Language

Two aspects of programming language suitable for scaling are language selection and implementation.

Each individual programming language has its unique strengths and weaknesses. Implementation of a system in a scaled manner affords the freedom to select an optimal language for the scaled system even though that language may differ from the one chosen for the target system. As an example, an assembly or machine-order language selected by necessity for a real-time message handling system may be replaced by a structured, higher-order language such as ALGOL or PL/I for the scaled version of that system. Such a selection might be made based upon considerations of top-down design, code readability, and modifiability, thereby contributing to accelerated program development.

With regard to programming language implementation, the language itself might be scaled. Consider a high-level, user-oriented interactive query language designed to implement a data base management system. Scaling might be accomplished by not implementing the query language at all in the scaled system (the functions would be provided by an experienced programmer), by implementing a subset of the language, or by implementing a version with cruder syntax which still supports the essential query requirements.

In the case of a compiled language implementation, an adaptation of scaled system methodology is in common practice today. When a compiler is developed, a compiler supporting a subset language is generally implemented first. Iterative enhancements of the baseline language are subsequently achieved through the use of the compiler itself to generate new compiler code. In this way, increased productivity is realized through the use of a higher-order language.

Another possible approach to language scaling would be in the case where firmware (PROM - programmable read-only memory) is to be employed in the final design. To speed the development effort, some functions to be ultimately supported by firmware might be implemented by software written in a high-level language.

Closely allied to functionality scaling, would be the choice to implement the support of a single compiler/language (COBOL, FORTRAN, PL/I, etc.) for a system which must support general purpose computing or to implement a single process-oriented language. An example of the latter might be a case where the functions of message editor and text editor would be supported by the system-standard editor for the scaled system.

8. Hardware Configuration

The choice of individual hardware components and their configuration is an important aspect of the scaled systems methodology. Significant savings in schedule, effort and cost may be achieved by reconfiguring the target systems hardware or by selecting an alternate operational environment for the scaled systems effort. A hardware configuration is the arrangement and number of physical system components that collectively comprise a system's "hardware", e.g., central processing units (CPU's), core memory, peripheral memory, input-output (I/O) devices, communications devices and the wired connections between them.

A hardware configuration may be scaled at the most elementary level by reducing either the number of component types or the total number of components. Either approach reduces total system complexity.

Some devices however, serve to reduce total system complexity by their presence and do not lend themselves to elimination for purpose of scaling. Examples would include intelligent terminals and peripheral controllers or I/O processors.

Consider the following list of feasible hardware modifications for scaling:

a. Reduce number of CPU's. A system which is ultimately to be multiprocessing could be scaled as a single processor.

b. The choice of CPU might, in fact, be different from that used for the full-scale system. (Conceivably, CPU choice could be an open question at the time the scaled system is developed.) The CPU used for the scaled system might be one with which the design team is familiar, one which is a substitute for a device under development or one which is "lesser" in terms of cost, capacity, speed or word size.

c. Number and/or type of peripherals.

d. Front-end/back-end systems could be scaled by preliminary work with only the front-end processor.

e. Increase the memory capacity of the scaled system. This could speed up the development effort by eliminating or reducing page faults and core swapping or reduce the need for overlays.

f. Reduce the complexity of interrupt handling. For example, use queued interrupts instead of prioritizing them. In the case of real-time systems, eliminate real-time interrupts by eliminating the input devices (simulate the data) or by considering them normal polled input devices.

Hybrid device types, complex devices, and in-development devices tend to increase system complexity and stretch out the development schedule. These could be avoided or replaced in the scaled effort. Problems with such devices can be reduced or eliminated by substituting existing, simpler, plug-compatible devices for them. Scaled effort, schedule, and cost can be reduced by replacing complex devices with simpler ones, in-development devices with existing ones, real-time devices by software simulation, and interrupt devices with processor-controlled ones. As noted, in some cases, it may be desirable to add hardware such as memory to reduce the degree of core utilization required or monitoring hardware to aid system evaluation, validation and verification.

Noteworthy is the fact that in many cases components do not have to be physically removed from a hardware configuration but merely logically disconnected or bypassed. Finally, considering the operational environment, it should be noted that multi-site and multi-national development facilities could be scaled by physically limiting the development to a single site, thus reducing complex communications requirements altogether.

With regard to communications between the processor and peripherals, the communications network itself may be scaled.

a. Reduce the number of nodes in the system.

b. Scale satellite communications with hard-wired, local systems.

c. Reduce the number of levels or complexity of a communications network or hierarchy.

d. Provide an equal level of service to each node rather than prioritized service.

e. Provide one-way rather than two-way message switching or communications.

f. Reduce the complexity of the logical hardware paths between the sender and receiver.

g. Provide a single communications path rather than include backup (alternate) links.

In the case of relatively small, firmware-based embedded systems such as on-board avionics systems, the facilities of a mainframe to reduce the need for a high level of machine utilization, to emulate I/O, and to support online, interactive program tracing and debugging would serve as an aid to the development of software for such projects. Again, in this case, "scaled" would not necessarily mean "smaller".

E. CONCLUSION

In order to derive a scaled system, it is necessary to take the functional specification of the full size system and apply some set of scale factors. These factors must be applied toward the purpose of simplifying the target system by a desired degree. The applications of the scale factors should be in accordance with a programmed set of objectives (not necessarily original design objectives) so that the scaling results in a useful product. The application of the scale factors to the functional specification should result in a scaled functional specification which becomes the master document for the design phase of the scaled system.

A summarized list of the possible scaling factors outlined in this report follows:

1. Data base
    a. Complexity and type of access method
    b. Complexity of data structure
    c. Size elements (number of files, length of files, etc.)
2. Performance
    a. Productivity/Throughput (system capacity, job mix).
    b. Responsiveness
    c. Utilization
    d. Operation System/Organization (processing mode, custom vs. existing OS, interrupts).
3. Functionality
    a. Vertical subsystem scaling (eliminate subsystem, utilities, etc.)
    b. Horizontal scaling (e.g., performance monitors).
4. Security
    a. File protection method.
    b. Dimensions of access matrix.
    c. Number of data sets.
    d. Classification level of users and/or terminals.
    e. Granularity of data access control.
    f. Number and types of access classifications.
    g. Codewords.
    h. Audit trail.

i. Authentication.

j. Encryption.

5. Maintainability

a. Process-error handling.

b. Restart/recovery.

c. Data correction.

d. Fault detection.

e. Monitors.

f. Backup.

g. Development aids.

h. Documentation.

6. Reliability

a. Precision.

b. Data error detection.

c. Approximation algorithms.

d. Coding standard enforcement.

7. Programming Language

a. HOL vs. assembly

b. Language subset

c. Single vs. multiple languages

d. Replacement of firmware

8. Hardware Configuration

a. Number and complexity of hardware

b. Number of CPU's

c. Type of CPU

d. Memory capacity

e. Interrupts

f. Hardware monitors

g. Number of communications nodes

h. Complexity of communication nework

i. Level of service to peripherals

APPENDIX C

SYSTEM SCALE FACTOR METRICS

24 December 1980

Peter Grimes

Dr. Marcia D. Kerchner

SYSTEM SCALE FACTOR METRICS

24 December 1980

Prepared by:

Peter Grimes
Dr. Marcia Kerchner

INCO, INC
7916 Westpark Drive
McLean, Virginia 22102

This report was prepared in support of contract F30602-80-0219 for the Rome Air Development Center (RADC), Griffiss AFB, Rome, New York 13441.

A. ABSTRACT

Scale factor metrics for each scale parameter are discussed and defined as an extension of the results of subtask 1.1 of the Scaled Systems research project. The research performed under subtask 1.2 of that project is summarized.

Scale factors are measures of the degree to which system parameters are scaled. Metrics are the unit measures chosen to express these system parameters. Using appropriate metrics, scale factors are defined in objective, quantifiable terms and in such a manner as to be indicative of their effects on cost, schedule, risk, and performance of scaled vs. full-scale systems.

B. OBJECTIVE

The objective of this research is to establish the basis through which current software engineering principles may be applied to the measurement of system attributes so that appropriate system scale factors may be systematically determined.

Scale factors relate a scaled system to its corresponding full-scale version based upon the criteria of cost, performance, and development schedule. Because this relationship is critical to forecasting and planning, it is important that it is based upon a sound analytical methodology and that it is accurately expressed.

C. DISCUSSION

1. Software Parameters

Through the research of software scale parameters, various aspects of software systems were identified as being relevant to scaling. These aspects include data base, performance, functionality, security, maintainability, reliability, programming language, and hardware configuration. Each aspect was subsequently broken down into its component parts. The list of software aspects and their component parts - collectively referred to as "software parameters" - formed the basis of this phase of the research.

2. Metrics

For each parameter, an attempt was made to identify a corresponding metric suitable for calculating scale factors. Very few parameters, however, could be expressed by existing metrics. The science of software metrics is still an infant discipline and there exist only a few software metrics generally accepted as such. These would include "Lines of Code" (LOC), "CPU works per second" (from Capacity Management principles), and "Manmonth" (or "manhour", "manday", "manweek", "manyear", or some equivalent).

3. Direct Metrics

This deficiency of currently available metrics, however, did not present a major obstacle to this phase of the research. This is due, in part, to the fact that many of the software scale parameters themselves imply a corresponding metric.

Consider, as an example, the case of data base size. Components of data base size include numbers of files, record types, and data field definitions, lengths of files, records, and fields. Each of these components describes its

own metric; the integral number of files is the metric for the "number of files" component, etc. The applicable scale factor is merely the value for the scaled system divided by the value for the full-scale system. This computation yields a percentage ratio - just like a scale ratio - that is conceptually attractive and easy to relate to and communicate.

4. Metric Indices

For software parameters that do not have a corresponding software metric and do not themselves imply the metric (e.g. complexity of access method, file protection method), the formulation of an appropriate scale factor is not as straightforward. In such cases a choice must be made between alternative scale factor formulation methodologies.

5. Interrelated Indices

One convenient alternative methodology involves the assignment of discrete metric values to each member in a group of related software attributes. Such metric values (or indices) could be assigned differently, depending on what they are related to. One possible method which has been rather extensively used involves interrelating the attributes with each other on a relative scale. An application of this scheme could be the factoring of the degree of file protection under the software aspect of security; "no file protection" would be placed at one end of the scale while "total file protection" would be placed at the other end, with the varying degrees of file protection falling in between. "No file protection" might be assigned a value of one and "total file protection" a value of three; thus, if no file protection were implemented on a scaled system emulating a full-scale system with a requirement for total file protection, the component scale factor would be computed as one divided by three, or 33%.

6. Global-Related Indices

A variation of this weighting scheme relates the component parameters to one or more of the principal global software system aspects - cost, schedule, risk, and performance. Consider these relationships:

Sequential file access methods would scale random access methods by reducing the inherent programming and data structure complexity resulting from the use of record dictionaries, links, and pointers. Substitution for such access methods, however, would also scale search time responsiveness (an element of performance) by a factor determined by the expected number of records present. These interrelationships will be studied in Task 2 of this project.

Another example arises in the enhancement of an operating system to support a particular application. The enhancements primarily provide ancillary functions - a basic scaled capability can be implemented without them. To obtain the source code to the operating system, become familiar with it, and modify it is costly and time-consuming; to retain the vendor to perform the modification is similarly expensive. The scale factor derived through the use of the "off-the-shelf" operating system can therefore be computed based upon the cost and schedule savings resulting from its use. Such a computation would probably be easier to formulate, communicate, and understand than attempting to determine a scale factor based upon the technical differences between the operating system and its modified version.

A point to keep in mind here is the motivation for scaling systems: achieving cost-effective system development with quality assurance. It is with this perspective that global-related scale factors are constructed.

It must be noted that this report does not purport to provide an authoritative definition of system metrics nor even scale factor metrics. Its intent, rather, is to assemble a preliminary set of metrics to provide a common discussion framework for scale factoring and a basis for subsequent research into the measurement and development of scaled systems.

Continuing research of software metrics will be beneficial to the scaled system project as well as the software engineering community through the ability to better quantify software system attributes. In an expanding discipline, definitions and emphasis tend to shift, contributing to the dynamic nature of the terminology and technical base. Actual metrics and relationships may therefore be re-sculptured as this project progresses toward the goal of achieving an understandable and workable methodology for scaled systems development.

D. TECHNICAL APPROACH

In determining scale factors, software parameters will be examined in the same order as they were reported in Software Scale Parameters, the report delivered under subtask 1.1 of this research effort and herein referred to as "Report 1.1".

For each aspect of software systems identified for scaling, a weight will be assigned to each full-scale function within the range of that capability. It will be determined what part of each function is implemented by the scaled system. The full-scale weights and scaled values are each added up and then divided to obtain the scale factor.

1. Data Base

a. Complexity and type of access method

There are three major data base access methods that are to be considered for scaling purposes: sequential access, indexed sequential, and direct access. Consider the average access times for a file of records using sequential and indexed sequential access.

$W_S = K_S' \cdot \frac{n}{2} = K_S n$

$W_I = K_I \log n$

Where K_S', K_S, K_I, Kare constants, W_S = average access time for sequential access, and W_I = average access time for indexed sequential access (includes table search time).

$$\text{ratio } R = \frac{W_S}{W_I} = \frac{K_S}{K_I} \frac{n}{\log n} = K\frac{n}{\log n}$$

For some $n=n_o$, R=1. That is, for a data base of n_o records, the sequential access and indexed sequential access methods yield the same search times.

The n_o could be determined by experimentation or experience.

For this R = 1, $\quad 1 = \frac{K n_o}{\log n_o}$

$$K = \frac{\log n_o}{n_o}$$

$$R = \frac{n}{n_o} \log n_o$$

Define R as the relative complexity.

For random access,

$W_r = ah + (1-a) K_r n$, where a = function a $(\frac{n}{s})$,
h = number of instructions in the hashing algorithm and s = size of the hashing region

ah is the hashing time and $(1-a) K_r n$ is the time to locate an empty space.

as $\frac{n}{s} \to 0$, $a \to 1$

as $\frac{n}{s} \to 1$, $a \to 0$

That is, as the region to which keys are hashed becomes denser, i.e., $\frac{n}{s} \to 1$, the random access method approaches the sequential method because the search for an empty spot will approach a sequential search.

The "cost" of a search can then be defined as

$C = C_I \cdot W_I + C_s \cdot W_s$, where C_I = Cost of an instruction,
C_s = Cost of storage, and W_I and W_s are the number of instructions and storage for a given method.

The "cost" calculations can be used as the scale metrics and can, for a given operating system, use the pricing algorithm of that particular system. The above formula is just one example of a costing algorithm. Another might be

$C = W_I \cdot W_s$.

The complexity of a given hierarchically structured data-base will be a function of the number of nodes, N, and the number of links between the

nodes, L.

For a hierarchical structure,

$N-1 \leq L \leq 2N - 3$

For a network structure,

$N-1 \leq L \leq \frac{N(N-1)}{2}$

The complexity metric will be of the form:

Relative complexity = $\frac{L}{N}$ links per node (record)

In the simplest case, with the only links being between parents and children, $C = \frac{N}{N-1}$ $\quad C \to 1$ as N increases

In the most complex,

$$C = \frac{2N-3}{N} = 2 - \frac{3}{N} \quad C \to 2 \text{ as } N \text{ increases}$$

In a network structure, the most complex case will be:

$$C = \frac{\frac{N(N-1)}{2}}{N} = \frac{N-1}{2} \simeq \frac{N}{2}$$

L/N can be viewed as an average number of links per node, where the more links a given node can have, the more complex is the implied structure. An absolute complexity might be defined as D= depth, the number of levels on the tree, or N, the number of nodes.

How a hierarchical data base is stored will be related to complexity, as well. Sequential listing of a tree structure is slow compared with linked list storage but the lists require extra storage and more complex programming.

The choice of storage organization for a network structure will result in the same variance. These interrelationships will be studied in a later phase of the project.

The relational representation of a data base is so simple that the measure of the complexity of any given relationally structured file would be a

linear function of the number of tables and rows. It can be thought of as a tree with each table representing a parent node at level 1 and the number of links equal to the total number of rows, the rows being on level 2 of the tree.

b. Complexity of data structure

(1) Hierarchical

The parameters that form a basis for scaling are the number of levels of the tree, the degree (number of successors to a given node), and the total number of nodes (records).

(2) Network

In the network model, it is also necessary to consider the linkage factor, where scaling would involve limiting the number of logical links between the nodes.

c. Size Elements

The elements of data base size lend themselves well to scale factoring. Because each is a quantum entity, the resultant scale factor may be computed as a fraction in which the value for the scaled system is represented in the numerator and the value for the full-scale system is found in the denominator. For example, the scale factor for "number of files" is found by dividing the number of files for the scaled system by the number of files for the full-scale version. Scale factors for the size parameters listed in Report 1.1 are as follows:

Parameter	Scale Factor
Number of Files	$\frac{\text{Number of Files (Scaled System)}}{\text{Number of Files (Full-Scale System)}}$

Length of Files (bytes)	$\frac{\text{Length of File (File - Scaled System)}}{\text{Length of File (File - Full-Scale System)}}$
Length of Records (bytes)	$\frac{\text{Length of Records (File - Scaled System)}}{\text{Length of Records (File - Full-Scale System)}}$
Number of Data Fields	$\frac{\text{Number of Fields (Records - Scaled System)}}{\text{Number of Fields (Records - Full-Scale System)}}$
Length of Data Fields (bytes)	$\frac{\text{Length of Field (Record - Scaled System)}}{\text{Length of Field (Record - Full-Scale System)}}$

2. Performance

The elements of performance suitable for scaling were identified in Report 1.1 as productivity, interactive responsiveness, utilization, and operating system organization.

a. Productivity

Productivity is a common measure of system performance. It is composed of two elements; the amount of work that can be physically accommodated and the rate at which it is ultimately accomplished. The first element is described by the system's capacity - the principal factor limiting workload. The second element is described by the systems throughput. Relating throughput to capacity yields an efficiency or performance index. Increasing system capacity implies acquiring additional hardware; increasing throughput, on the other hand, entails obtaining a corresponding increase in the operating system's efficiency, although this can also be accomplished through new hardware.

(1) Capacity

Borrowing from Capacity Management technology, a system's capacity may be defined as the amount of information it can contain at any certain period of time. The metric used to measure information is the byte (eight Boolean bits or the equivalent of one alphanumeric character), and these are aggregated for each external device type and for the total internal memory to arrive at the system's "capacity", the total number of bytes in the system.

(2) System Power

System power can be derived if the rate at which it can manipulate information (bits or bytes) between the various capacity components can be determined. The number of bytes, or amount of information, systems can manipulate internally and between peripherals in a given amount of time is generally a known quantity, and thus can be used as the metric. By collecting and correlating this type of information, one can begin to determine the relative power of different systems by comparing their capacity and ability to handle data. Cost is directly correlated with power. Scaling systems can thus be achieved through scaling their power at the cost of not being able to store and process as much data at a given time or at as fast a rate.

(3) Hardware Capacity

Hardware components provide metrics by which they may be measured and compared. Memory size is a good example as is the speed of a communications line. A one megabyte memory module scales four megabytes by 75% (the resultant scale factor is 25%); a 300 baud modem scales a 3600 baud one by 92% with a resulting scale factor of 8%.

(4) Software Capacity

Each element of the software can, again, provide its own metric, e.g. table size can be scaled by reducing the number of bytes. The number

of bytes metric would also apply to input and output field sizes. Robustness of a system, the ability to handle a broad spectrum of data volumes in excess of that originally anticipated, can be scaled by implementing a minimum of error checking. The metric would be number of error conditions to be checked in the system.

Throughput is a measure of the system s efficiency of using resources. Throughput is usually a function of the operating system but is not restricted to such and it is generally expressed as the amount of work processed in a certain time frame. This can be best visualized by considering a batch-type environment; the metric would be defined as: Number of user jobs completed/ unit time, the more user jobs completed in a given amount of time the greater the throughput. Similarly, the more job-steps completed in a given amount of time the greater the throughput. In addition, input data rates are a measure of throughput. When throughput is related to capacity, a performance/efficiency index is obtained. In general, throughput is a function of a large number of factors including the percentage of time that a system is operable. They are all intimately related to productivity and, when combined, yield a measure of performance efficiency. Scaling performance has considerable potential for cost savings because realizing high efficiency in EDP systems tends to drive costs exponentially and schedules proportionally higher.

b. Interactive Responsiveness

Interactive responsiveness was defined in the proposal as the inverse of response time, that is, the number of responses/unit time. This definition maintains consistency in defining parameters so that their "down" direction implied scaling and their "up" direction implied unscaling. Responsiveness is dependent on many factors. In a message switching system, responsiveness is dependent on such parameters as message control efficiency, communication line

speeds, and the number of message transceivers present; in an information based system, it is dependent upon keyed-search efficiency, storage unit access times, retrieval speeds, frequency of queries, etc.

It should be mentioned that responsiveness is very difficult to predict on the front-end of the implementation phase. This parameter is usually quantified through observation. In the past, response times were typically established as a system requirement. If the resulting system did not meet the target, much work was expended to modify the system and bring the response time within specifications. Research has shown that this is consistently the most costly way to effect what essentially are design changes - on the tail-end of the development cycle. Scaled system development, on the other hand, provides a scaled model of the ultimate system which would conspicuously reveal such design deficiencies and the full-scale system design specification can be cost-effectively adjusted in the front end of the design cycle - where economic leverage is the greatest. Suppose it is anticipated that a system's responsiveness will degrade in direct proportion to the number of terminals connected to it. If a scaled system with one-tenth as many terminals does not respond in less than the targeted response time, it should be clear that there exists a deficiency in the design specification. Although simplified, this is probably a typical analysis example for responsiveness.

c. Utilization

The effects of high processor utilization on costs and schedule are fairly well documented; above approximately 50% utilization, costs begin to rise exponentially and schedules grow proportionally - 90% utilization will triple the costs of 50% utilization. Such figures have generally been derived from studies concerned with core memory and processor time utilization.

The computation of utilization is fairly straightforward, as the differences can be attributed to the entity under scrutiny. Utilization can be measured in terms of the percent capacity used; alternatively, it can be measured according to the time used as compared to the time available. If necessary, the utilization of each component of the system can be measured. For example, CPU utilization would be measured in instructions per time, memory utilization as proportion of total memory available, etc. As an example of the calculation, an eighteen megabyte disk/drive containing nine megabytes of information is described as being 9/18, or 50% utilized. Similarly, if a Management Information System (MIS) package is on-line for a total of six hours during an eight hour workday due to user demand, its utilization may be computed as 6/8, or 75%.

d. Operating System/System Organization

Report 1.1 identified subelements of Operating System/System Organization as processing mode, operating system, and interrupt processing. In that report, the difficulty in quantifying these aspects was addressed. Scaling system aspects applicable under this category would undoubtedly be highly case-dependent and quantifying the factors largely subjective.

(1) Processing Mode

In a system where batch, interactive, and real-time processing modes are supported, a scaled system could consider only a single mode of operation. Also, a real-time system could be scaled with a batch system and simulated input data. Measuring the resulting decrease in complexity (if measuring could be done at all!) would appear to be not as valid as investigating resultant changes in other, quantifiable system aspects which interrelate with the processing mode.

(2) Operating System

As noted in report 1.1, if the full-scale system requires a custom operating system, the scaled system could use an off-the-shelf system or modify an existing one. The resultant cost and schedule changes can be used as the metric.

3. Functionality

H.D. Mills states that the basic functions (60-80% of the processing) of a unit of software are usually a small fraction (20-40%) of the total software finally built. This assertion is generally accepted and holds deep implications for Scaled Systems Technology. Since effort is strongly correlated with produced code, an initial operating capability of a full-scale system (barring ancillary functions, documentation, installation, maintenance, and user support) could be achieved with only 20-40% of the total projected effort. This attests to the viability of the scaled systems approach and identifies functionality as a principal system aspect suitable for scaling. Functionality can be scaled by reducing the variety of functions supported (eliminating ancillary or additional support functions), or by reducing functional complexity. The first method entails vertical functional scaling (eliminating sub-systems); the second - horizontal functional scaling.

a. Modularity

When speaking in terms of functionality, modularity is probably the system parameter that is being most directly dealt with. Modularity describes the number and composition of the various program modules comprising a system. The complexity metric of a system can be defined in a manner analogous to that used for defining the complexity of a hierarchical structure.

Absolute complexity = number of modules

Relative complexity = $\frac{\text{number of module linkages}}{\text{number of modules}}$

b. Factoring Vertical Functional Scaling

In the case of vertical scaling, scale factors could be computed based solely upon the number of functions eliminated as compared to the total number of functions called for in the requirements or design specification (3 of 12 functions eliminated reduces functionality by 3/12, or 25%; the scale factor would subsequently be computed as: (12-3)/12, or 75%).

Preferably, the amount of code necessary to support each function would be a known quantity. Thus, if the three functions discussed in the previous example required 40,000 lines of code (LOC) from a total system size of 100,000 LOC, the resultant scale factor would be 60% ((100,000-40,000)/100,000), as opposed to 75%.

The absolute complexity could be scaled by reducing the number of modules. A more accurate metric for measuring the scale factor might be number of lines of code.

c. Factoring Horizontal Functional Scaling

Deriving a scale factor for horizontal scaling may be more difficult. In the case of eliminating a common shared functional module, such as a monitor or security subsystem, the analysis could be analogous to that of vertical scaling. If, however, horizontal scaling is achieved by reducing module sizes due to decreased complexity, the analysis may have to be more subjective. As in the case of operating system/system organization, computation of horizontal scale factoring will be reserved for a case-by-case analysis and future research.

For a message handling system, the receipt/transmission can be scaled by omitting some of the functions. For example, a full-scale system might have message receipt, transmission, dissemination, storage, and retrieval capabilities while the scaled system might receive messages from only one input source, not transmit messages, etc. The scale factors for these functions would be defined as the percentage of full-scale functionality implemented by the scaled system.

For message-receiving systems, the number of networks with which the system interfaces could determine the weight factor in the metric:

Scaling message receipt factor =

$$\frac{\text{number of network interfaces in scaled system}}{\text{number of network interfaces in full-scale system}}$$

The functions of message transmission, e.g. handling new messages and retransmission, imply a weight assignment in this case of two to the full-scale system.

{Transmission, dissemination} =

$$\frac{\text{number of \{transmission, dissemination\} functions in scaled system}}{\text{number of \{transmission, dissemination\} functions in full-scale system}}$$

The weights of the full-scale system and those of the scaled system factors can be summed to produce a total functionality scale factor:

$$\text{Functionality scale factor} = \frac{\Sigma \text{weights in scaled system}}{\Sigma \text{weights in full-scale system}}$$

As more detail about the full-scale system design is acquired, it will be possible to further refine these metrics, by assigning weights to the proposed subfunctions that reflect the complexity or resource requirements for implementation. At this level of detail, typical metrics used previously include source lines of code estimates, staff estimates, and number of pages

devoted to each function within the functional description.

4. Security

Currently, system security is a topic of prime consideration and vigorous research. As more information is entrusted to computer systems, concern for security necessarily increases. This concern has been manifested in such areas as automatic encryption technology and operating systems through the Kernelized Secure Operating System. To date, many security strategies have been implemented including the use of multiple processor networks to distribute varying levels of classified material and to ensure that the determinancy of access privileges can be maximized.

Barring automatic cyphering/decyphering hardware for communications, much of computer security is achieved through overhead software. This may be accomplished at any one of the many system levels: kernel, executive, (operating) system, sub-system, and application. Since few operating systems are built with the goal of providing information processing with multiple security levels, most security schemes are implemented at the sub-system level or below. Regardless of system level, security processing primarily involves the validation of resource requests against tables cross-referencing valid requestors (users and programs) and resources. Such tables require maintenance modules to keep them up to date as well as access and search modules (which must also be secure!). Depending on the degree of security provided, these tables grow increasingly complex and their associated processing overhead grows; thus, comparing the magnitudes of such tables provides an adequate means of quantifying security metrics in addition to the obvious metrics of required design and coding effort. Additionally, many systems identify the need for access audit modules which track details concerning requests for resources and the security modules' resulting actions.

The security probability (Gilb) is defined as

P (a) = probability of successful attack rejection

This probability will vary depending on the level of protection in effect.

a. File protection

Six levels of file protection have been defined by Randall Jensen in Software Engineering, as follows, with a scale value attached to each:

Scale Value	Levels Of File Protection
1	no protection- File access and all operations available to any user
2	all or nothing- If access granted, then all operations permitted
3	controlled sharing- User is granted access rights which are the minimum necessary to accomplish the specified task
4	specified access- Access to each object is restricted and access rights are owner-definable; the rights could be based on the user's identity, the environment (type of terminal, time of day, etc.), or the contents of a record
5	total protection- No file sharing at all
6	post-access control- Users granted access subject to the purpose for which it is to be used after access accomplished

b. Dimensions of access matrix

The dimensions of the matrix are determined by the number of users, processes, or procedures which have access restrictions and in the other direction by the number of objects for which access is restricted. These two factors to be scaled would have metrics as follows:

$$\frac{\text{number of users with access restrictions}}{\text{total number of users}}$$

$$\frac{\text{number of procedures with access restrictions}}{\text{total number of procedures}}$$

c. Number of data sets

Different data sets could be provided for each access clearance.

d. Classification Level of users and/or terminals

To scale classification level, all terminals/users could be granted access to everything and provide only physical security for access to the terminals. The estimated amount of effort necessary to implement a subsystem (LOC) which would allow other than open access would be used as the metric.

e. Granularity of data access control

Access to different user classes could be modified.

f. Codewords

Codewords could be eliminated, with the metrics as follows:

Ø = No Codewords
1 = Codewords

g. Audit Trail

A similar metric could be defined as follows:

Ø = No audit trail
1 = Audit trail

h. Authentication approach

Each function, including the use of passwords, the recording of access failures, log-on procedures, and terminal authentication, would be assigned a weight determined by the amount of software (lines of code) needed to implement it. The scaling resulting from the simplification or elimination of these capabilities could be measured by the percentage of lines of code eliminated.

i. Encryption

Through the use of simulated data or limited transmission, a full-scale requirement for software encryption of data could be suspended. The metric would be:

0 = No encryption
1 = Encryption

As more detail about the proposed system is acquired, the weights of {0,1} could be adjusted to indicate in some way their complexity or resource requirements for implementation. However, for simplicity, the present metrics will only indicate whether or not the function is implemented.

5. Maintainability

Building-in maintainability is generally done to minimize the time required to locate and fix a bug in the software during the test, integration, and maintenance phases of its life-cycle. This time will inevitably be directly proportional to the amount and quality of supporting technical documentation available. Of course, complementary aspects of maintainability are the auto-correcting, recovery, or diagnostic facilities supplied with the final software product. Scaling such aspects of maintainability as amount of documentation and maintenance aids supplied may seem contrary to sound developmental practices but it must be remembered that the anticipated life-cycle of a scaled system is short (just long enough to get an effective "handle" on the design and functionality of the full-scale system); the scaling of these aspects can therefore be justified.

The appropriate metrics suitable for factoring maintainability are amount of documentation produced and functionality of the ancillary maintenance modules (the auto-correcting, recovery, and diagnostic software - See "Functionality"). In addition, the effort required for configuration management may be scaled in the respect that the configuration management necessary for the scaled effort need not be as extensive as that of a full-scale system. Again, consideration of the expected life-cycle duration is paramount.

Maintainability is defined as the probability that, when maintenance

action is initiated under stated conditions, a failed system will be restored to operable condition within a specified time t.

Maintainability is a function of the capabilities included in the system, the skill level of the personnel, and the support facilities (locally available tools and diagnostic test equipment or aids, spare parts/ alternative program versions/back-up files). It is a measure of the cost and time required to fix software errors in an operational system. Among the maintenance modules which could be scaled are:

a. Process error handling

The scaling involved in minimizing the number of conditions to be checked can be measured by the number of lines of code needed to implement the error checking.

b. Restart/recovery procedures

The restart procedures can be eliminated as much as possible and a set of values assigned as follows:

Ø = no restart procedures
1 = minimal restart procedures
2 = complete restart procedures

c. Data correction

Modules to correct and/or reject bad data can be eliminated with the scaling being measured by the reduction in the number of lines of code.

d. Fault detection

Fault location/trap software can be eliminated, reduced, or modified, again using the number of lines of code as the metric.

e. Monitors

Software to monitor system performance and gather statistics can be eliminated and the (estimated) number of lines of code needed to implement these functions can be used as the metric.

f. Backup

Backup procedures can be minimized.

g. Development aids

Development aids such as program tracers and interactive debuggers would actually be added, to reduce the development effort of the full-scale system.

h. Documentation

Documentation objectives are concerned with the quality and quantity of user publications. Scaling the amount of documentation may be risky as pointed out in 1.1, because the lack of documentation, recovery, and reconfiguration programs may actually hamper rather than enhance the program.

6. Reliability

Reliable software is software that does not fail. The metric commonly used for reliability is the frequency of failures occurring over a specific period of time. Obviously "building-in" high reliability is costly and is only justified in applications demanding infallible software, such as man-rated applications (i.e. applications where lives are at stake). One software aspect reflecting upon reliability is robustness. Robustness describes the software's ability to adequately accommodate erroneous input values; values which, if undetected, could cause the software to produce inappropriate output, fail, or "crash". The pitfall of error detection is the ability to anticipate all input combinations which could cause the software to fail. This requires rigorous requirements formulation and design; reliability cannot be "tested" into software. As such, reliability is a worthy area for scaling and again the scaled system approach presents the opportunity to validate and refine the typically heuristic systems formulated for error detection and correction.

The concept of reliability is contrasted to that of maintainability in that maintainability is concerned with readily fixing or enhancing the software whereas the trust of reliability is to prevent failures from occurring in the first place. Accordingly, a certain amount of redundancy is built into systems such that automatic diagnosis and recovery can be accomplished by the software itself without operator attention or intervention. Such methodologies are a principal component of data base management systems where the need for the ability to detect a degraded data structure and rebuild it are crucial to their reliable operation. This redundancy requires additional design, system storage, programming, and effort; and as such reliability may be scaled with respect to these aspects.

Reliability can be defined as probability of satisfactory performance for a given time when used under stated conditions, the metric being defined as the number of failures/time. A software error is present when the software does not do what the user reasonably expects it to do. A software failure is an occurrence of a software error.

a. Precision

The precision metric is defined as the number of decimal places or bits, whichever is the most convenient unit to use for the particular application.

Data error detection

Software geared to errors which would appear infrequently in practice or not at all in the input to the scaled system can be eliminated.

The metric will be defined as the number of lines of code for error detection.

c. Approximation Algorithms

Scaling can be accomplished through the use of fast, easy (not as accurate as possible) approximation functions and algorithms.

The motivation for scaling approximation algorithms is to minimize lines of code or complex operations which are prone to error.

The lines of code required to implement algorithms could be the metric. The logical complexity of a program, a measure of the degree of decision-making within a system, could also be used. The absolute logical complexity measure is defined as the number of non-normal exits from a decision statement (IF,ON,AT END, etc). The relative logical complexity is defined as:

$$\frac{\text{Absolute Logical Complexity}}{\text{Total number of Instructions}}$$

To minimize complexity, maximize the independence of each component of a system.

d. Coding Standards

Relaxation in enforcement of coding standards would only be done in cases where recoding would be necessary to implement the full system. If, however, the scaled system will form the basic structure for the full system, then strict coding standards should be maintained.

7. Programming Language

The important principles in language syntax and semantics are uniformity, i.e. a language construct that appears in several contexts should have the same syntax and semantics and simplicity, which implies clarity and integrity of language concepts.

More often than not, the choice of a programming language is set or, at best, limited at any one development installation. Selection of an alternate language can be prompted by a number of reasons. These include non-existence

or limited support of the target machine or language and development complexity of the target language.

Consider the case of "ADA", the proposed DoD standard programming language which, at the time of this writing, has been specified but not yet fully implemented. A software development installation could still begin work on a project targeted for ADA-language implementation through the use of an existing high-level language, designing and coding it with the anticipation and intention of future conversion to ADA. While it may be difficult to visualize the "scaling" in this example, it represents the use of software other than that targeted so that a preliminary product can be readily assembled and evaluated for any design or operational deficiencies with the intent of minimizing the overall development schedule, risk, and cost.

In contrast to no language, there may be no machine available for the development of a software application. This case is not infrequent, as software projects are often started in anticipation of the delivery of hardware (which is invariably delivered late), or the production of hardware which is not yet marketed but whose characteristics have been fully specified. In these cases, the software project need not be delayed, as the tools of cross-assemblers, compilers, hardware simulators and emulators can be utilized so that the scaled production, evaluation, and design iteration can get underway.

The writing of software, much like any other creative process, is largely an iterative process involving the refinement of working "drafts" toward the goal of a product in final form. Some languages facilitate this type of process more readily than others even though they may not be the best choice for the ultimate implementation. A perfect example of this is that of the language interpreter. Typically (and necessarily) slow and inefficient in terms of execution speed and run-time hardware requirements, language interpreters are interactive

and promote and facilitate quick implementation of anything from an application program to an operating system through built-in type checking, syntax analyzers, statement editors, and break-pointing.

"Structured" languages are also well suited to subsequent modification and semantical re-working while languages resembling assembler dialects are more difficult to read and comprehend and thus harder to use in iterating towards a software solution. This is one of the underlying aspects of structured, high-level languages and part of the reason they contribute to shorter development schedules and increased programmer productivity. An assembly language-based application can be scaled with respect to programming language through the choice of a high-level language to work out the basic logic of the application in a structured manner. After design validation, the chore of language conversion to assembler for code optimization is relatively straightforward. This is analogous to arguments presented in favor of simulation languages, which have been used quite successfully in many different instances. Such a methodology would be ideal for the development of software intended for embedded applications and is, in fact, a common practice in the development of such software as avionics and hand-held devices such as programmable calcualators and language translators.

Report 1.1 cited other instances for scaling language selection and implementation. Scaling language implementation is accomplished by successively enhancing a base-line subset of the language being implemented - an iterative enhancement technique which is similar to the scaled approach.

Even though scaling programming language is feasible, the factoring of this aspect is difficult; much research, however, has been devoted to quantifying the relative expressive powers of languages. Perhaps the best known work

of this type is that of Halstead's Software Science. Through the basic tools of software science, Halstead was able to develop a methodology for factoring the expressive power of languages on a scale. Further discussion of Halstead's work here would be a digression, the point being that programming languages have been analyzed and assigned ratings as to their relative "power". In the context of scaled systems, such a rating could be used to imply a measured impact on development effort of typical applications. Additional data is available quantifying the expressive power of languages at the machine level, this being the expansion ratio of machine instructions to high-level language statements. Halstead, Knuth, and others have made contributions in this area.

8. *Hardware Configuration*

As in the case of data base, factoring hardware configuration is simplified by the nature of the entity itself, due to the numerically descriptive nature of hardware. Hardware is basically described by its capacity, transfer rate, quantity, and cost, where the basic scale factor definition would be:

$$\text{Scale factor (in \%)} = \frac{\text{value (metric) for scaled version}}{\text{value (metric) for full-scale version}}$$

Consider the following list of hardware elements possible for scaling and their metrics:

a. Number of CPU`S

Scale from multiprocessing to a single processor

Processing scale factor =

$$\frac{\text{number of CPU`s in scaled version}}{\text{number of CPU`s in full-scale version}}$$

b. Number of Peripherals

Peripheral scale factor =

$$\frac{\text{number of peripherals in scaled version}}{\text{number of peripherals in full-scale version}}$$

c. Instruction set of a CPU

Instruction set scale factor =

$$\frac{\text{number of elements in scaled instruction set}}{\text{number of elements in full-scale instruction set}}$$

It must be noted, however, that some devices serve to reduce system complexity by their presence. Examples would include intelligent terminals and peripheral controllers or I/0 processors.

d. For some factors, simulation could be used to reduce complexity in the scaled system. For example, eliminate real-time interrupts by eliminating the input devices (simulate the data instead).

e. Regarding communications between the processor and peripherals, the number of communications nodes could be easily factored by the standard definition.

Communication node scale factor =

$$\frac{\text{number of communication nodes in scaled version}}{\text{number of communications nodes in full-scale version}}$$

f. Complexity of communications network or hierarchy

The complexity can be scaled by lowering the number of linkages among nodes.

g. Level of service to peripherals

A scaled system could provide an equal level of service to each node node rather than prioritizing service. The scaling would be based on the complexity associated with prioritized service.

Parameter	Metric
1. Data Base	
a. Complexity of access method	cost metric
b. Complexity of data structure	
(1) relative	links/node for relational, $\frac{R}{R+T}$, R=number of rows, T=number of tables
(2) absolute	number of levels, number of nodes
c. Size elements	number of files, length of files (bytes), length of records, number of fields, length of data fields, (bytes)
2. Performance	
a. Productivity/throughput	
(1) System capacity	total number of bytes in system
(2) System power	number of bytes/time
(3) Hardware capacity	Σ of capacities of individual components
(4) Software capacity	number of bytes in tables, etc; number of error conditions to be checked in the system
b. Interactive responsiveness	number of responses/unit time
c. Utilization	capacity used/capacity available, time used/time available
d. O.S./Organization	
(1) Processing mode	number of modes of operation
(2) Operating system	cost and schedule changes
3. Functionality	
a. Modularity	absolute complexity = number of modules relative complexity = number of module linkages
b. Vertical subsystem scaling	number of lines of code
c. Horizontal functional scaling	LOC, staff estimates, number of pages of functional description
4. Security	
a. File protection	levels of file protection {1-6}
b. Dimensions of access matrix	number of users with access restrictions/total number of users number of procedures with access restrictions/total number of procedures
c. Number of data sets	
d. Classification level of users and/or terminals	LOC for a classification-subsystem

Parameters	Metric
g. Granularity of data access control	Minimize for each level of access
f. Codewords	Ø = no code words 1 = codewords
g. Audit trail	Ø = no audit trail, 1 = audit trail
h. Authentication	LOC
i. Encryption	Ø = no encryption, 1 = encryption
5. Maintainability	
a. Process-error handling	number of conditions to be checked
b. Restart/recovery	LOC
c. Data correction	LOC
d. Fault detection	LOC
e. Monitors	LOC
f. Backup	
g. Development aids	
h. Documentation	number of pages
6. Reliability	
a. Precision	number of decimal places or bits
b. Data error detection	LOC for error detection
c. Approximation algorithms	LOC required to implement algorithms, absolute and relative logical complexity
d. Coding standard enforcement	
7. Programming Language	"power"
8. Hardware Configuration	
a. Number and complexity of hardware	number of CPU's, number of peripherals, number of elements in instruction set
b. Interrupts	
c. Complexity of communications	number of communications nodes number of linkages among nodes network
d. Level of service to peripherals	complexity of prioritized service

APPENDIX D

INTERRELATIONSHIPS AMONG SCALING FACTORS

21 January 1981

Dr. Marcia D. Kerchner

INTERRELATIONSHIPS AMONG SCALING FACTORS

21 January 1981

Prepared by:

Dr. Marcia D. Kerchner

DYCO, INC
7916 Westpark Drive
McLean, Virginia 22102

This report was prepared in support of contract F30602-80-0219 for the Rome Air Development Center (RADC), Griffiss AFB, Rome, New York 13441.

INTERRELATIONSHIPS AMONG SCALING FACTORS

The scale factors proposed in Report 1.2, System Scale Factor Metrics, influence and interrelate with each other in complex ways that can be quite different for different operating regimes (e.g., disk-limited, CPU-limited) of the IDHS being modeled. In order to scale a system, ways are needed of predicting changes in system characteristics as the scaled parameters vary, even when the variations are large enough to place the IDHS into a different operating region. For example, if the small scale system has a factor of four fewer terminals than the envisioned full-scale system, it is necessary for the system designer to know how system throughput will degrade when the system is scaled up and terminals are added.

In many engineering applications, the amount by which critical parameters vary is in some sense "small", and it is possible to represent the relationships as linearized expansions about some nominal operating point. Unfortunately, the kind of scaling that is appropriate in the present application is generally characterized by variations ranging from a factor of 2 to 10. IDHS, when scaled by these magnitudes, will often be operating in entirely different regimes, and no simple expressions relating the performance characteristics in different regimes can generally be constructed.

As an illustration, consider the functional relationship of system throughput to a scaled parameter such as CPU power for an IDHS operating in a disk-limited or a CPU-limited operating regime.

Consider Q, the ratio of the length of time the CPU is occupied to the length of time the disk is occupied:

$$Q = \frac{\frac{\text{CPU instructions}}{\text{disk access}} * \frac{\text{seconds}}{\text{CPU instruction}}}{\frac{\text{seconds}}{\text{disk access}}} = \frac{\text{seconds CPU occupied}}{\text{seconds disk occupied}}$$

Where $\frac{\text{CPU instructions}}{\text{second}}$ is the CPU power.

When Q is less than one, the system is disk-limited. Figure 1 demonstrates how such a disk-limited system might look with two jobs running, with control of the CPU and disk alternating over time. The jobs generally finish using the CPU quickly and must wait for the slower disk.

When the system is disk-limited it tends to be rather insensitive to CPU speed, but throughput varies greatly with changes in disk access time and with those software changes, e.g., in data base organization, that vary the CPU instructions executed per disk access. That is, the behavior of a disk-limited system (most jobs are in the disk queue) is sensitive to:

- Disk hardware characteristics (e.g., speed, size)
- Data base organization affecting disk accesses per search
- Security features that require disk accesses
- Available main memory where this influences paging and/or swapping rates

The behavior is insensitive to:

- CPU power
- Software changes that affect the number of computational instructions
- Data base organization that doesn't affect disk accesses (e.g., file size in a random access configuration)

TIME	JOB 1 STATUS	JOB 2 STATUS
↓	Job 1 gets CPU	Job 2 in CPU queue
	Job 1 gets disk	Job 2 gets CPU
	Job 1 continues using disk	Job 2 in disk queue
	Job 1 gets CPU	Job 2 gets disk
	Job 1 in disk queue	Job 2 continues using disk

Figure 1. Job Behavior in a Disk-limited System

On the other hand, if the quantity Q is larger than one, as illustrated in Figure 2, most jobs end up waiting for CPU services and the system is more sensitive to changes in CPU power and those parameters that affect the number of instructions.

The behavior of a CPU-limited system (most jobs are in the CPU queue) is sensitive to:

- Changes in CPU power
- Software changes that affect the number of computational instructions.

The behavior is insensitive to:

- Speed, size of disk hardware
- Overhead features such as security, that require extra disk accesses to perform specific functions
- Data base organization affecting disk accesses/search
- Available main memory

The behavior described in these examples of CPU-limited and disk-limited systems is summarized in Figure 3, which illustrates the throughput and CPU speed functional relationship. It shows, for example, that doubling CPU power does not necessarily double throughput.

The relationsh ps between parameters are complex and non-linear. It is not possible t· write down analytic expressions that will hold under all conditions.

In order to provide the system designer with the tools that will enable him to predict performance under the wide range of scaling conditions that are encountered in practical situations, a concept has been evolved that uses a simulation model of a generalized intelligence data handling system to predict performance and to predict changes in one variable from changes in another. In effect then, the simulation

TIME	JOB 1 STATUS	JOB 2 STATUS
↓	Job 1 gets CPU	Job 2 in CPU queue
	Job 1 continues using CPU	Job 2 in CPU queue
	Job 1 gets disk	Job 2 gets CPU
	Job 1 in CPU queue	Job 2 continues using CPU
	Job 1 gets CPU	Job 2 gets disk
	Job 1 continues using CPU	Job 2 in CPU queue

Figure 2. Job Behavior in a CPU-limited System

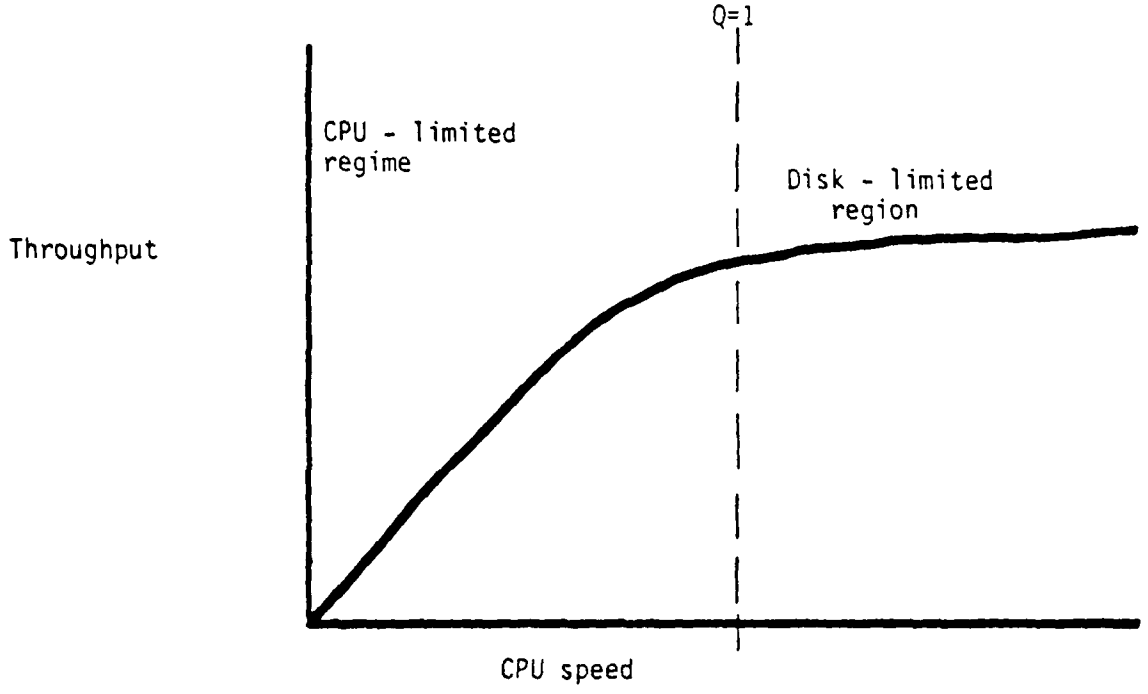


Figure 3. - Throughput - CPU speed functional relationship

substitutes for the nonexistence of precise analytic functional relationships between various scaled parameters.

It has been found that a fairly small number of parameters is adequate to specify each particular IDHS to the simulation. Each of the input parameters, in turn, can be expressed as a fairly simple analytic function of the scaling parameter factors. A series of formulae are used in steps to relate the simulator variables to scale factors. A diagram of the technique is shown in Figure 4. An example of a simulator input variable is CPU service time, i.e., time in CPU per CPU block, where a block is a set of instructions until a disk access is encountered. The following formulae are one set that can be used to relate CPU service time to system scale factors.

$$\text{CPU service time} = \frac{\text{instructions executed per block}}{\text{power (instructions per time)}}$$

$$\text{Total instructions executed} = \text{number of computational instructions} + \text{number of disk accesses} * \left[\frac{\text{security instructions}}{\text{access}} + \begin{array}{l}\text{other}\\ \text{overhead}\\ \text{instructions}\end{array} \right]$$

$$\text{Number of disk accesses} = \text{number of data base accesses} + \text{number of paging accesses}$$

$$\text{Number of paging accesses} = K1 * \text{number of instructions executed} * \frac{\text{virtual core per job}}{\text{real core per job}}$$

$$\text{Real core/job} = \frac{\text{hardware core} - \text{operating system core} - \text{security core} - \text{maintenance core}}{\text{number of terminals}}$$

The above step-by-step procedure to relate simulator variables to scaling parameters, such as number of terminals and security and maintenance core (as functions of the levels of protection and maintainability required), is illustrated in Figure 5 for CPU and disk

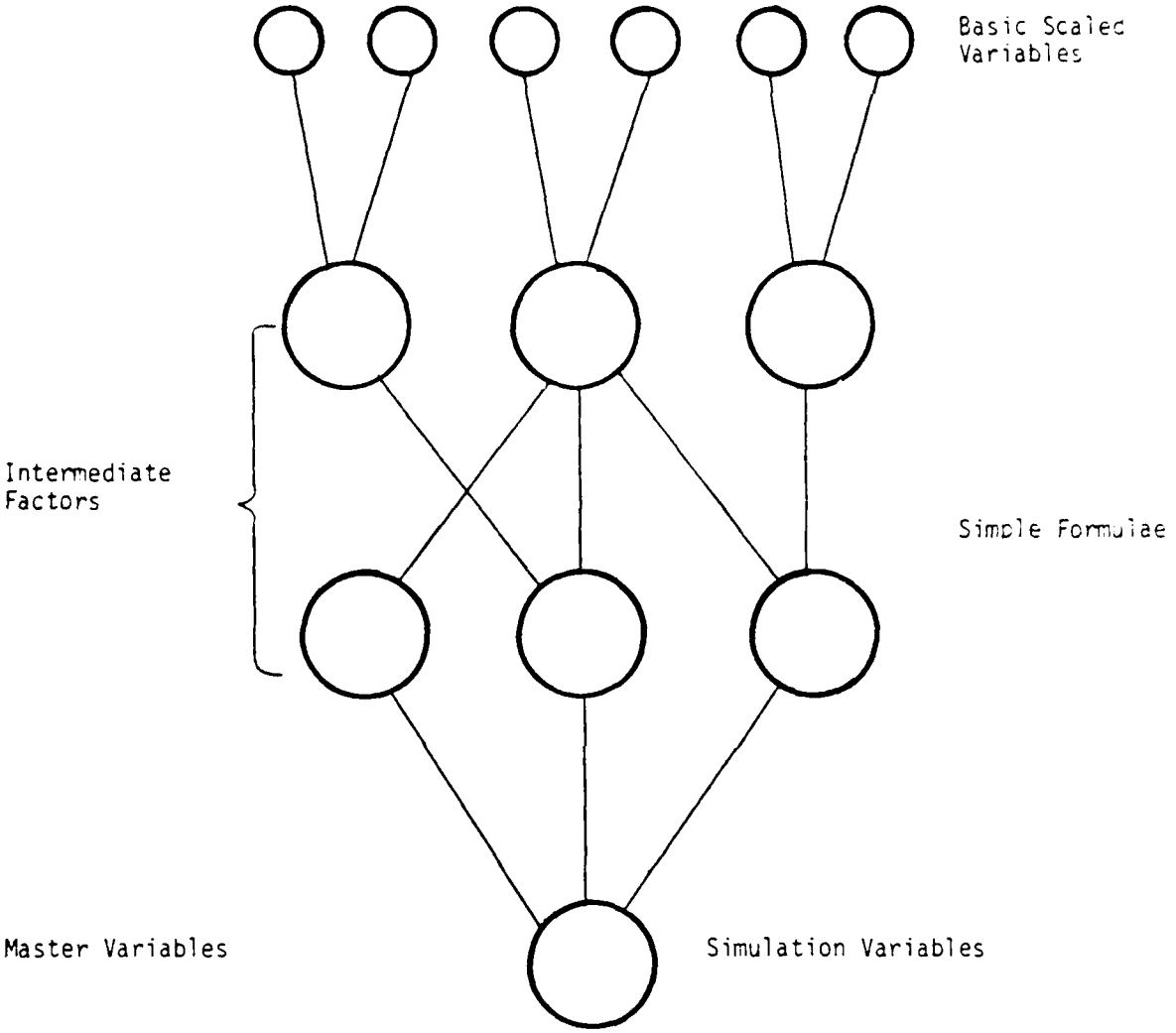


Figure 4. Relating Scale Factors to Simulation Variables

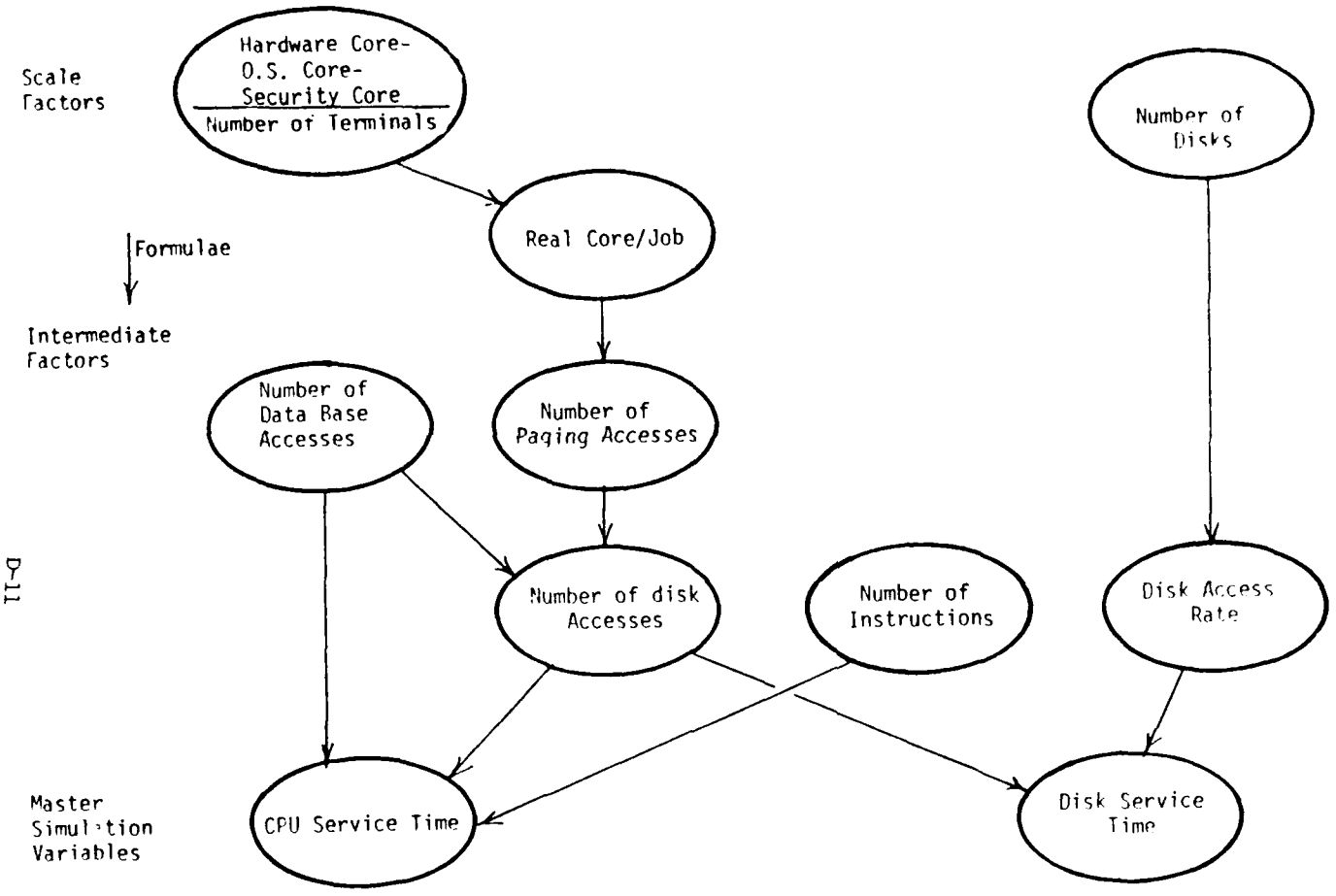


Figure 5. Relating Scale Factors to CPU and Disk Service Time

service time.

Other parameters of which throughput is a function include disk service time, job rate (number of jobs input per time), number CPU requests per job, system power, and system capacity.

Several of the elements in these formulae, e.g., paging accesses, can be measured by the system, and the constants such as K1 can be derived through system measurements.

Unknown parameters, e.g., the number of computational instructions for a typical job, must be evaluated in order to complete the formulae. A way to approach the problem of evaluation might be to start with "reasonable" estimates for these parameters. When the small scale system is operational, they can be measured by monitoring system behavior. Indeed, the purpose of building the small scale system is to measure the parameters which will be used in the full-scale system so that flexibility in the design of the full-scale system can be retained. The small scale system together with the simulation will enable the designer to see what will work in the full-scale system.

The simulation will be used by the system designer in an iterative manner in the course of specifying the full-scale system. The scaling factors will be specified and used as input to the simulation, the output will be examined, and scaling will be respecified until the desired outputs, i.e., full-scale system behavior are achieved. A typical question would be: How much can the data base size be scaled up with present disk hardware without going below the minimum required responsiveness (responses per unit time)? Will it be necessary to have more and/or faster disks in order to achieve the desired full-scale

system responsiveness and incorporate the necessary data base size? If access time is improved by so much, how much can the data base then be scaled up? The system designer will look at the results of the simulation based on a set of values for the scaling parameters and iteratively adjust these values. Such respecifications of scaling may well result in design changes for the full-scale system, e.g. by going to more and/or more powerful hardware. Thus the tools to be used will be the simulator and the set of input variables.

The remaining research on this task will involve further definition of the method's details and insuring that the simulation has sufficient realism for the case of IDHS. In addition, it must be verified that the functional relationships between the scale factor parameters, as measured by the defined metrics, and the simulation input variables are valid relations. If necessary, metrics will be redefined.

APPENDIX E

SIMULATOR VARIABLE - SCALE FACTOR EQUATIONS

27 February 1981

Dr. Marcia D. Kerchner

SIMULATOR VARIABLE - SCALE FACTOR EQUATIONS

27 February 1981

Prepared by:
Dr. Marcia D. Kerchner

INCO, INC.
8260 Greensboro Drive
McLean, Virginia 22102

This report was prepared in support of contract F30602-80-0219 for the Rome Air Development Center (RADC), Griffiss AFB, Rome, New York 13441.

Simulator Variable - Scale Factor Equations

The report Interrelationships Among Scaling Factors described a procedure to relate simulator variables to scaling parameters. The definition and equations have been refined and will be described.

Consider the simulation parameter CMEAN, mean CPU service time. It can be defined as follows:

$$\text{CMEAN} = \frac{\text{instructions executed/block}}{\text{instructions/time (power)}}$$

Consider also the following definitions:

N_D = number of disk accesses

= N_{DB} (number of data base disk accesses) +

N_{DP} (number of paging disk accesses)

I = number of instructions

= I_C (number of computational instructions) +

I_{DB} (number of data base instructions) +

I_P (number of paging instructions)

Define the frequency of data base disk accesses per computational instruction,

$$F_{DB} = \frac{N_{DB}}{I_C} .$$

Then $N_{DB} = I_C[\frac{N_{DB}}{I_C}] = I_C F_{DB}$.

Also, $N_{DP} = K_P^* \ I_C * \frac{C_V}{C_R}$, where K_P is a system-dependent constant calculated as the number of paging accesses/computational instruction, C_V is the virtual core for a particular job (the job size), and C_R is the real core for the job (the actual core available for the job).

Then

$$N_D = N_{DB} + N_{DP}$$

$$N_D = I_C[F_{DB} + K_P \frac{C_V}{C_R}]$$

$$\frac{I_C}{N_D} = \frac{1}{F_{DB}+K_P\frac{C_V}{C_R}}$$

$\frac{I_C}{N_D}$ is the number of instructions per block so,

$$CMEAN = \frac{[F_{DB} + K_P \frac{C_V}{C_R}]^{-1}}{\text{instructions/time}}$$

To find a value for F_{DB}, estimates and typical numbers will be sought. The value will depend on the function being performed and the probability of having to make a disk access. There are several factors that affect the probability that a piece of information is in core vs. on disk, such as the amount of the data base that is stored in core at any time, the organization of data on the disk (the data base structure), and the data manipulation algorithms. Also, since F_{DB} was defined as $\frac{N_{DB}}{I_C}$, it may be possible to calculate N_{DB} for a given function and data base organization, while I_C would also be a function of scale factors, such as the function being performed and the data base size and data base complexity. Thus F_{DB} could be derived in this way.

To find C_V, for each job of type j, assume input values for simulation parameters mean $C_V(j)$, $C_V(j)$. As the job begins, pick the actual C_V according to a probability distribution function.

For C_R, the real available core for the job, the following system-dependent values can be input:

C_T = total core for the machine

C_{OS} = operating system core

Then

$$C_R = \frac{C_T - C_{OS}}{\text{Number of jobs running}} = \frac{C_T - C_{OS}}{\text{Number of terminals}}$$

Now consider the simulation parameter, DMEAN, the disk service time.

DMEAN = seek time + disk read speed*average amount read.

The seek time is a function of hardware, a scale factor, and usually dominates DMEAN. Whether the other element, disk read speed*average amount read, is negligible or not depends on how the system is handled.

Another simulation parameter IMEAN, is defined as follows:

IMEAN = CPU/disk iteration count

= number of disk accesses

= $N_D = N_{DB} + N_{DP}$.

Then

$$IMEAN = I_C[F_{DB} + K_P \frac{C_V}{C_R}].$$

If F_{DB} is difficult to calculate, the following equation can be used instead:

$$IMEAN = N_{DB} + I_C * K_P \frac{C_V}{C_R}.$$

In the equations that have been discussed, the simulator parameters have been defined as functions of many of the scale factors, including power (instructions/time), number of terminals, real core (system capacity), number of instructions (related to data base complexity & structure), hardware, functionality, and security core (involved in the calculation of C_R, the real available core for a job). The use of the simulator with experimental values will then permit analysis of scale factor interrelationships.

APPENDIX F

SIMULATOR DESCRIPTION

27 February 1981

Dr. Marcia D. Kerchner

Simulator Description

27 February 1981

Prepared by:

Dr. Marcia D. Kerchner

INCO, INC.
8260 Greensboro Drive
McLean, Virginia 22102

This research paper was supported by the United States Air Force contract number F30602-80-C-0219 for the Rome Air Development Center (RADC), Griffiss Air Force Base, Rome, New York 13441.

Simulator Description

The operating system performance simulator operates as follows: a job enters the system at random intervals from one of n terminals. The job is assigned a job class (disk or CPU bound) and a CPU/disk iteration count based on a probability distribution.

The job is placed on the CPU or disk queue if the required facility is busy; when it gains control of the CPU, it is assigned a CPU service time based on a probability distribution function; similarly, a disk service time is assigned when it gains control of the disk. When the job has been completely serviced, the terminal that submitted the job waits a period of time based on a user-submitted probability function until a new job is submitted from that terminal.

Another method of describing the simulation is by enumeration of its elements, i.e., its objects, terminals, jobs, CPU, and disk, as shown in Figure 1, and its events as shown in Figure 2.

Events can be job creation, job start, CPU event, or disk event. The description of each is shown in Figure 2.

Objects:

- n terminals
- Jobs
- CPU
- Disk

Characteristics of objects:

- Terminal:
  - wait time
  - number of terminals
  - active job
- CPU:
  - queue
  - active job
  - service time slice
- Disk:
  - queue
  - active job
  - service time slice
- Job:
  - status - CPU queue, CPU active, Disk queue, Disk active, completed
  - Class
  - CPU/Disk iteration count

Figure 1. Simulation Characterization of Objects

Job creation event

1. creates a job object
2. assigns a job class & iteration count randomly
3. schedules next job creation event based on user creation rate

Job start event

1. activates job by placing it in CPU queue
2. schedules CPU event if CPU queue is empty

CPU event

1. If job has CPU, determine if it is finished. If finished:
    a. delete job object
    b. remove job from terminal
    c. schedule a job start event

    If not finished:
    a. add job to disk queue
    b. if disk free, schedule disk event
2. Assign next job in CPU queue (if any) to CPU.
3. Determine time slice for this CPU slice.
4. Schedule CPU event for this time.

Disk event

1. If job has disk:
    a. add job to CPU queue
    b. if CPU free, schedule CPU event
2. Assign next job in disk queue to disk.
3. Determine disk time slice for this disk access.
4. Schedule disk event.

Figure 2. Description of Simulator Events

APPENDIX G

PRELIMINARY RESULTS OF SIMULATION INVESTIGATION OF SCALE FACTOR INTERRELATIONSHIPS

15 April 1981

Dr. Marcia D. Kerchner

Preliminary Results of Simulation Investigation of Scale Factor Interrelationships

15 April 1981

Contract No. F30602-80-0219

Prepared for:
Rome Air Development Center
Griffiss AFB
Rome, New York 13441

Prepared by:
Dr. Marcia D. Kerchner
INCO, INC.
8260 Greensboro Dr.
McLean, VA. 22102

The operating system performance simulator has been exercised with various sets of test data for two purposes. First, to examine how scale factors interrelate in a given environment, and second, to demonstrate how the simulator would be used in an actual implementation situation.

As was pointed out in the report entitled "Interrelationships Among Scaling Factors", the relationships between system parameters, i.e., scale factors, are complex and non-linear. It is not possible to derive analytic expressions that will hold under all conditions. The simulation model of a generalized intelligence data handling system can be used to predict performance and to predict changes in one variable from changes in another. The simulation thus substitutes for the nonexistence of precise analytic functional relationships between various scaled parameters.

The test data set was designed to enable system performance to be evaluated for different combinations of parameter values that permit comparative analysis of system scale factor interrelationships. Some of the issues addressed include how the number of terminals, the mix (the combination of CPU-bound and disk-bound jobs) and average CPU service time, affect disk waiting time, CPU and disk utilization, response time, and other measures of system performance.

The simulator selects CPU and disk service times and terminal wait times using Poisson distributions. This distribution models arrivals in a very satisfactory fashion.

The value of examining different values for a parameter such as CPU service time is that it is a way of simulating added overhead that features such as security operations may require. Extra disk accesses

may also be required to perform specific security functions, so individual implementations of security systems will affect system performance in varying ways, as a function of these application-dependent system parameters (CPU time and disk accesses).

The test runs indicate that the number of terminals appears to be a prime factor in system performance, while adding CPU overhead, such as security features, does not change response time significantly. As shown in Figure 1, plotting the number of terminals against the average disk wait demonstrates that the job mix and number of CPU/disk iterations play very little role in the resultant average disk wait; regardless of whether the system is CPU- or disk-bound, the average disk wait increases almost proportionately with the number of terminals, e.g., the disk wait with 16 terminals is approximately twice the disk wait with 8 terminals.

As shown in Table 1, in a system of 8 terminals, with 50% of the jobs CPU-bound, the average disk wait is 235 time units. When 67% of the jobs are CPU-bound, the average disk wait is 217 time units. Even when the system is made more heavily CPU-bound, with 67% of the jobs in this category, and an average CPU time slice approximately half of the disk time slice, the average disk wait for 8 terminals is 207, compared with 236 for a 67% CPU-bound system with CPU time slices only 3.5% of the disk time slice. No dramatic changes in disk wait time have taken place from changing the job mix values. Similarly, doubling the average number of CPU/disk iterations does not have much impact on the average disk wait, as demonstrated in Table 2. However, the response time doubles as the number of CPU/disk iterations doubles, a consideration for those overhead operations requiring disk accesses.

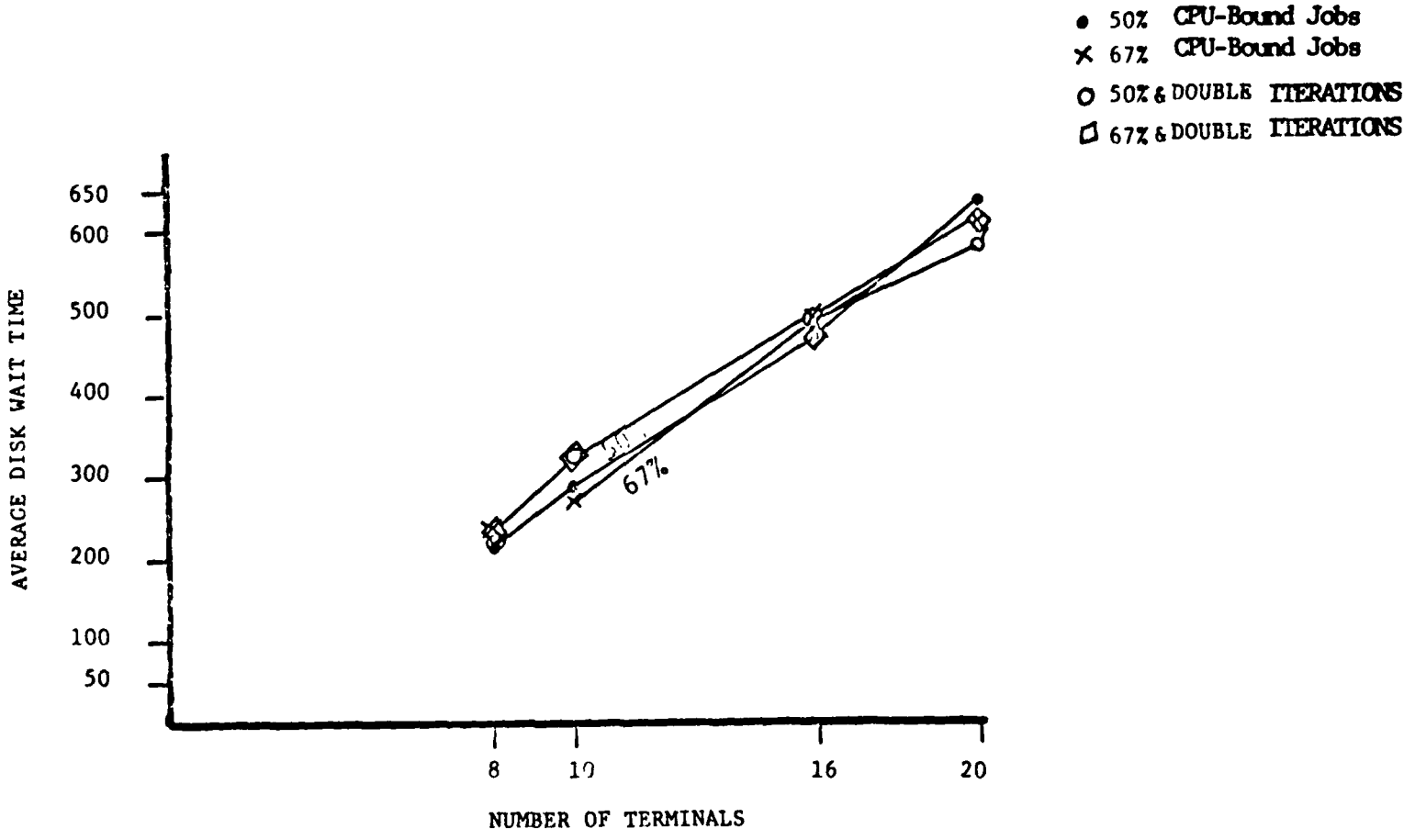

• 50% CPU-Bound Jobs
× 67% CPU-Bound Jobs
○ 50% & DOUBLE ITERATIONS
□ 67% & DOUBLE ITERATIONS
AVERAGE DISK WAIT TIME
650
600
500
400
300
200
100
50
50%
67%
8
10
16
20
NUMBER OF TERMINALS

Figure 1

No. of Terminals	Average Disk Wait	Average CPU Wait
8	235	1.96
10	305	1.76
16	477	2.22
20	621	1.92

50% Job Mix

No. of Terminals	Average Disk Wait	Average CPU Wait
8	217	1.65
10	279	1.71
16	458	2.17
20	618	1.85

67% Job Mix

Table 1: Average Disk & CPU Waits

Double CPU/Disk Iterations

No. of Terminals	Average Disk Wait	Average Response Time
8	240	2323
10	308	2763
16	477	4009
20	609	4930

50% Job Mix

No. of Terminals	Average Disk Wait	Average Response Time
8	223	1910
10	297	2301
16	447	3396
20	623	4218

67% Job Mix

Base CPU/Disk Interations

No. of Terminals	Average Disk Wait	Average Response Time
8	226	1128
10	303	1457
16	477	2040
20	632	2788

50% Job Mix

No. of Terminals	Average Disk Wait	Average Response Time
8	223	1194
10	250	1064
16	468	1812
20	612	2233

67% Job Mix

Table 2: Average Disk Waits & Response Times for Base & Doubled CPU/Disk Interations

Disk utilization is consistently over 99% regardless of the variations in the values for the parameters. This result is to be expected due to the fact that most computer systems will be limited by the nature of the disk hardware, i.e., its speed.

CPU utilization remains at about 2% to 3% when the average CPU time slice is approximately 3.5% of the average disk time slice and increases to 15-25% when the CPU time slice is increased to approximately half that of the disk time slice. Thus, it is difficult to come anywhere near loading the CPU.

Figure 2 shows how the response time reacts to changes in the job mix. As might be expected, response time is lowest when the largest percentage of jobs is CPU-bound, i.e., the 67% curve. The rate of change in response time as the number of terminals increases can be seen to be fairly consistent. The three upper curves plot the response time resulting when the number of CPU/disk iterations is doubled.

It is valuable, too, to examine what happens when the terminal wait time or "think time" is approximately doubled. This factor relates to job rate (the number of jobs input per time). Table 3 summarizes the results of test runs which indicate that there is very little change in the response time when the wait time is doubled; sometimes, it increases a bit, sometimes it decreases, and sometimes it does not change. Thus, it would appear that terminal wait time can be changed within reasonable limits without significantly affecting response time.

It must be kept in mind that the parameter values used in the simulator for these experiments are just one attempt at approximating a real system and an example of how the simulator would be used under real

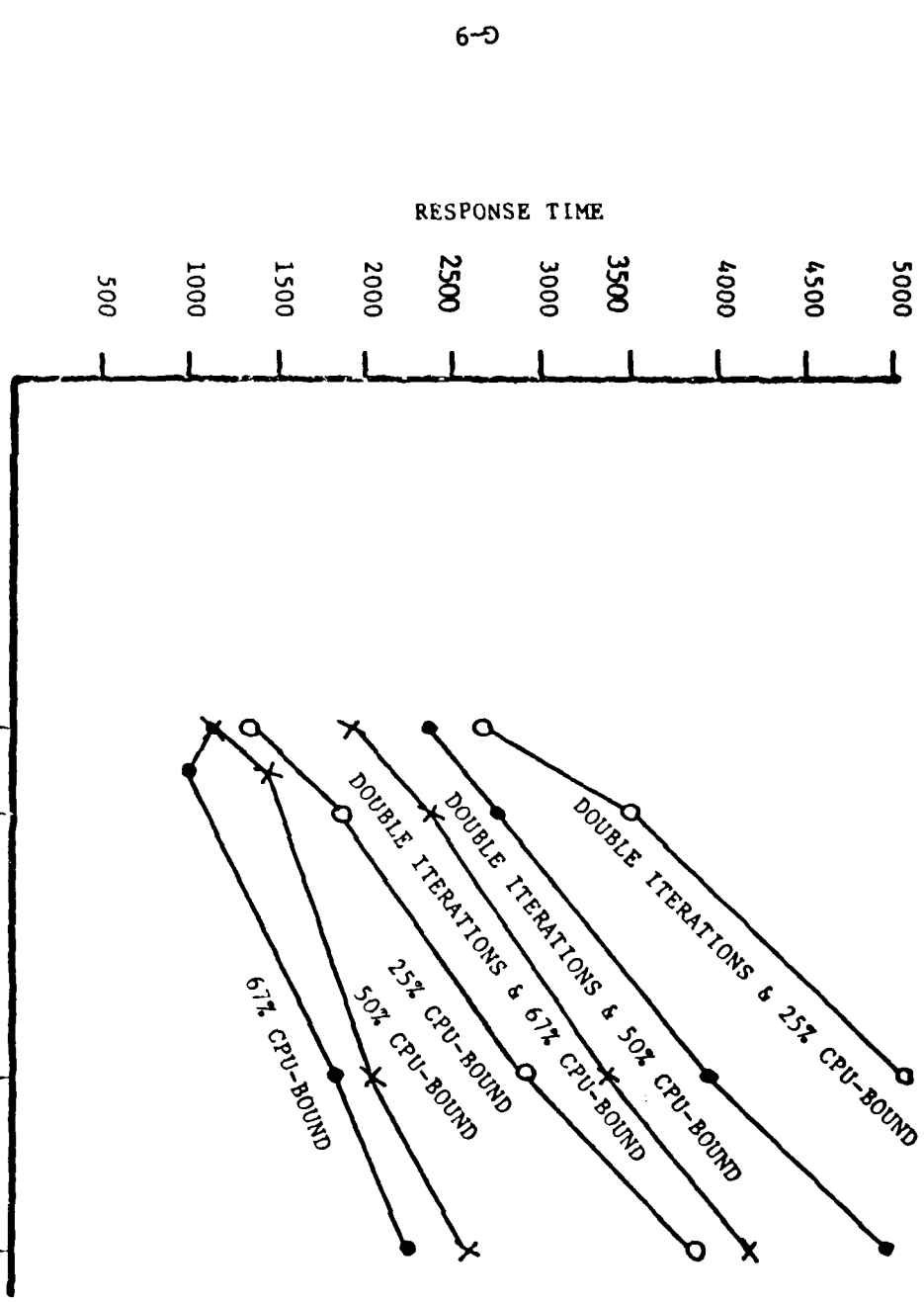

RESPONSE TIME
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
8
10
16
20
NUMBER OF TERMINALS
DOUBLE ITERATIONS & 25% CPU-BOUND
DOUBLE ITERATIONS & 50% CPU-BOUND
DOUBLE ITERATIONS & 67% CPU-BOUND
25% CPU-BOUND
50% CPU-BOUND
67% CPU-BOUND

Figure 2.

No. of Terminals	Response Time for 1 Wait Time	Response Time for Double Wait Time
8	911	856
	1691	1934
10	1025	1067
	2485	2112
16	1728	1966
	3559	3249
20	2327	2241
	4784	4733

Comparison of Response Times When Wait Time Doubles

Table 3

circumstances. That is, conclusions have been reached based on tests involving, for example, 20 vs. 10 terminals. These conclusions might not hold when one is considering scaling 100 terminals to 50 terminals. Recent expansion of the simulator's capabilities has made possible experiments with a larger number of terminals up to a maximum of 100, permitting the examination of interrelationships in that operating region. The results of these tests will be described in a later report.

The value of the present results, however, is in pinpointing those scale factor interrelationships that should receive attention when scaling is required.

AD-A110 867 INCO INC MCLEAN VA F/G 9/2
SMALL SCALE SYSTEMS.(U)
SEP 81 M KERCHNER, P GRIMES F30602-80-C-0219
UNCLASSIFIED INCO/1155-681-TR-46-D(F) RADC-TR-81-251 NL

APPENDIX H

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Aron, J.D., "Estimating Resources for Large Programming Systems," NATO Conference, Rome, 27-30 October 1969.

2. Basili, Victor R. and Zelkowitz, Marvin V., "Analyzing Medium-Scale Software Development," Proceedings Third International Conference on Software Engineering, 10-12 May 1978.

3. Basili, Victor R. and Zelkowitz, Marvin V., "Operation of the Software Engineering Laboratory," Second Software Life Cycle Management Workshop, U.S. Army Computer Systems Command, August 1978.

4. Basili, Victor R., "A Meta-Model Software Development Resource Expenditures," Department of Computer Science, University of Maryland.

5. Basili, Victor R., "Resource Models," Unpublished manuscript, Department of Computer Science, University of Maryland.

6. Beizer, Boris, "Micro-Analysis of Computer System Performance," Van Nostrand Reinhold, New York, 1978.

7. Doty, D.L.; Nelson, P.J.; Steward, K.R., "Software Cost Estimation Study Guidelines for Improved Software Cost Estimating," RADC-TR-77-220, Vol. I and Vol. II, August 1977.

8. Freiman, Frank R.; Park, R.E., "Price Software Mode-Version 3, an Overview," IEEE/Poly Workshop on Quantitative Software Models, IEEE Press No. TH0067-9, New York, 9-11 October, 1979.

9. Horowitz, Eillis, et. al., Practical Strategies for Developing Large Software Systems, Addison-Wesley Publishing Co., Reading, Mass., 1975.

10. IEEE Proceedings, Workshop on Quantitative Software Models, IEEE Publishing Co., New York, NY, 1979.

11. IWTS Technical and Operational Reference Manuals, INCO, INC., October 1979.

12. Junk, McCall, Putnam, and Walters, "Survey of Software Cost Estimating Techniques," GE TIS 78CIS010, 17 May 1978.

13. Lasher, William, "Software Cost Evaluation and Estimation: A Government Source Selection Case Study," IEEE/Poly Workshop on Quantitative Software Models, October 1979.

14. NMIC Functional Analysis/Enhancement Study Final Report, prepared under government contract #F30602-80-C-0145, INCO, INC., December 1980.

15. Norden, Peter V., "Project Life Cycle Modeling: Background and Application of the Life Cycle Curves," Software Life Cycle Management Workshop, Airlie House, 21-23 August 1977.

16. Parr, F.N., "An Alternative to the Rayleigh Curve Model for Software Development Effort," IEEE Transactions, pp 291-296, May 1980.

17. Putnam, Lawrence H., Software Cost Estimating and Life-Cycle Control: Getting the Software Numbers, IEEE Press No. EHO 165-1, New York, October 1980.

18. Putnam, Lawrence H., "A General Empirical Solution to the Software Sizing and Estimating Problem," IEEE Tutorial Software Cost Estimating and Life-Cycle Control, IEEE Press No. EHO 165-1, New York, October 1980.

19. Putnam, Lawrence H., "Progress in Modeling the Software Life Cycle in a Phenomenological Way to Obtain Engineering Quality Estimates and Dynamic Control of the Process," IEEE Press No. EHO 165-1, New York, October 1980.

20. Putnam, L.H., Fitzsimmons, Ann, "Estimating Software Costs," I, II, III, Datamation September-November 1979.

21. Putnam, Lawrence H., "SLIM," QSM Sales Literature, 1979.

22. Reference Manual - PRICE Software Model, RCA PRICE Systems, Cherry Hill, New Jersey, December 1977.

23. Runyan, Linda and Schatz, Willie, "Applications Development," Datamation, March, 1981, pp. 165-166.

24. Walston, C.E. and Felix, C.P., "A Method of Programming Measurement and Estimation," IBM System Journal, Vol. 16, No. 1, 1977.

25. Wolverton, Ray W., "The Cost of Developing Large-Scale Software," IEEE Transactions on Computers, Vol. 23, No. 6, 1974.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.


ATE
ME