AD-A047 674 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO F/G 9/2
A QUANTITATIVE ANALYSIS OF ESTIMATING ACCURACY IN SOFTWARE DEVE--ETC(U)
AUG 76 P F GEHRING
UNCLASSIFIED AFIT-CI-77-28 NL

1 OF 2
AD A047674

A QUANTITATIVE ANALYSIS OF ESTIMATING ACCURACY IN SOFTWARE DEVELOPMENT

A Dissertation

by

PHILIP FRANCIS GEHRING, JR.

Submitted to the Graduate College of
Texas A&M University
in partial fulfillment of the requirement for the degree of

DOCTOR OF PHILOSOPHY

August 1976

Major Subject: Computing Science

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CI 77-28 ✓	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Quantitative Analysis of Estimating Accuracy in Software Development		5. TYPE OF REPORT & PERIOD COVERED Dissertation
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Philip F. Gehring, Jr		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT Student Texas A&M University, College Station TX		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/CI WPAFB OH 45433		12. REPORT DATE August 1976 13. NUMBER OF PAGES 178 Pages
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited

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

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE AFR 190-17.

JERRAL F. GUESS, Captain, USAF
Director of Information, AFIT

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

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

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(14) AFIT-CI-77-28

(6) A QUANTITATIVE ANALYSIS OF ESTIMATING ACCURACY IN SOFTWARE DEVELOPMENT.

(9) Doctoral thesis,

A Dissertation

by

(10) PHILIP FRANCIS GEHRING, JR.

Approved as to style and content by:

Udo W Pooch
(Chairman of Committee)

Preston C Ellis
(Head of Department)

Roger W Eckhardt
(Member)

Robert J Anderson
(Member)

Eugene B Smith
(Member)

(12) 191p. (11) August 1976

012 200

ABSTRACT

A Quantitative Analysis of Estimating

Accuracy in Software Development

(August 1976)

Philip Francis Gehring, Jr.

B.S., U.S. Naval Academy

M.S., Georgia Institute of Technology

Chairman of Advisory Committee: Dr. Udo W. Pooch

This research quantitatively examines the estimating accuracy of over 5000 standardized resource consuming activities from 39 software development projects of various size, which were accomplished at the U.S. Air Force Data Systems Design Center. The activity data pertaining to planned hour estimates and actual expenditures were collected by an automated project management system (PARMIS) as the data were generated.

The dissertation hypothesizes that specific activities can be isolated which consistently have a greater influence on whether a software development project will be successful in terms of cost and schedule estimates. The arithmetic and percent differences between estimated and observed hour expenditures are the elementary variables used to investigate estimating accuracy. Various summarizing and statistical techniques are employed to reveal the information inherent in the data, and to identify, if possible, a correlation between the selected activities and the final difference between the total hours estimated and expended for the project. The findings from the data source used clearly support the hypothesis. However, no correlat as found between the activities which have the most influence o ting accuracy in a software development project and other crit such as the total project difference. The

primary conclusion of this work is that software estimation is still very poor and inconsistent because the existing model for software development and traditional estimating techniques are incompatible. A new development model is described in addition to a recommendation for a centralized, standardized, software project management system which would service several different agencies developing different types of projects.

ACKNOWLEDGEMENTS

I must first acknowledge the love, patience and understanding given to me by my family while I was pursuing this degree. Dr. Udo W. Pooch, my committee chairman, has alternately listened to my problems, guided, counseled, and pushed me over the many hurdles associated with my studies and research. More than my advisor he has become a trusted friend and a passable racquetball adversary. Appreciation is due to my committee members, Dr. Roger W. Elliott, and Dr. Robert J. Anderson of the Computing Science Division, and Dr. Eugene B. Smith of the Business Analysis Department. Dr. James R. Scoggins has been most considerate as my Graduate College Representative. I am also indebted to Major William Greene, USAF, for assisting with editing the dissertation, to Christopher Gehring for acting as my research assistant in the preparation of tables, and to Geoffrey Gehring for keypunch support. Mrs. Linda Schwierzke typed and retyped all draft material and was most responsive. My friend Mr. Eugene Kelly of the Air Force Data Systems Design Center deserves much credit for his technical help and responsiveness in providing the data base for this research.

DEDICATED TO

MY MOTHER AND FATHER

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The Problem

It is generally accepted in both the professional and academic worlds that, at present, it is almost impossible to accurately estimate the time and cost of the development of the software for new automated data systems [86]. Very few, if any, completed major software development projects have met four recognized criteria for success.

1. Preliminary cost estimates must be accurate within reasonable bounds.
2. Production schedules must be met.
3. The operational requirements and design performance criteria must be achieved.
4. The product must be reasonably error free and fundamentally dependable when installed in an operational environment [105, 136].

The objective of producing reliable software is certainly not at odds with the need to reduce the high cost of software development [53].

Previous research into the problem of accurately forecasting software development has suffered from overly ambitious objectives. This has often been the fault of the research sponsors who want to be able to demonstrate a solution to a problem justifying the expended research money. For example, in the middle 1960's the U.S. Air Force spent hundreds of thousands of dollars to solve the problem of accurately estimating software development. What they received for their money was a set of equations designed to assist in estimating span days, man hours, etc. for programming and testing computer programs [82]. The research agency promptly disclaimed the accuracy of these

---

The Communications of the ACM is used as a pattern for format and style.

equations for various reasons. The foremost reason was inadequate data which had to be collected by questionnaire after the development projects had been completed. Attempting to leap from the existing meager knowledge of how to accurately estimate the cost of the software development process, which is complex to say the least, and little understood by management and technician alike, to an equation which will accomplish the task, is overly optimistic and not very scientific.

What is required is basic research, accomplished in small steps, which will reveal some of the underlying idiosyncrasies about the software development process. As with all research, each discovery made will provide more insight into the development process and the concomitant estimation problem and create a broader foundation of knowledge for further study. The necessary data are beginning to be collected in management information systems used to control software development projects [30]. These systems themselves and the data contained therein need to be scrutinized and studied. Do they contain the proper kind of data for research? What do the data themselves reveal? Under what kind of organizational structure and management environment was the data collected? How disciplined was the project planning and data collection and reporting? What assumptions does the system rely on? Software engineering simply must know more about itself before that discipline shall be provided some gestalt tool (in terms of handy equations) which will resolve its estimating and cost dilemmas.

Alfred Pietrasanta has concluded that:

> Many of the problems of resource estimating are symptoms of an underlying ignorance of the process of program system development for which the estimates are being made. The serious students of estimating must first be willing to probe deeply into the fascinating and complex system development process; to uncover the phases and functions of the process; to highlight the subtle interrelationships of the program system being developed and the project organization doing the developing.
>
> To the fainthearted, such a study can be self-defeating, because it will uncover dozens of factors that

> influence estimates and will lead to the conclusion that the process is overwhelmed by unpredictable variability. Estimating can then be considered an exercise in futility.
>
> Those who persevere, however, will recognize that examining the influencing variables and their causal relationships is precisely what is required if estimates are ever to be improved. Only then can we do meaningful quantitative research and scientific analysis of resource requirements. We are never likely to eliminate unpredictable variability, but we should be able to go a long way toward improving predictability far above today's primitive state-of-the-art. [116]

Wolverton sagaciously observes that the software industry is young, growing and marked by rapid change in technology and application. It is not surprising, then, that the ability to estimate costs is still relatively underdeveloped [158]. As true as this statement may be, there exists an extensive bibliography on the subject matter, as demonstrated by the 1,500 entries at MITRE Corporation [30]. There does not seem to be any problem with admitting that there is a problem. At the 1968 NATO Software Engineering Conference, Professor Edsger Dijkstra observed that the general admission of the existence of software failure, in that group of responsible and renowned people, was the most refreshing experience he had had in years; the admission of shortcoming being the primary condition for improvement [106]. The admission notwithstanding, improvement in estimating software development has not been very rapid or very significant. Some of the conditions contributing to the limited understanding of software development and inability to predict the time and resources required in this development have been listed by Bratman:

1. Lack of discipline and repeatability.
2. Lack of development visability.
3. Changing performance requirements.
4. Lack of design verification tools.
5. Lack of software reuseability. [20]

It is also interesting to note that there is a pessimistic forecast regarding improvements in software production into the 1980's in even the more recent studies and literature [78, 30]. In fact, progress in computing in the future will probably be severely strained by the need for more accurate and dependable methods for estimating and controlling development costs and schedules.

Software and the Development Process

This research will examine the problem of accurately estimating the time of some of the activities accomplished in the development of new software. In the context of this work software will mean any computer program or module thereof. A program is a set of transformations and other relationships over sets of data and container structures (records, words, sections, etc.) [106]. Software is used in its broadest sense, and may include operating system programs, utility programs or application programs. A software development project may encompass the design, coding, testing, etc. of a single program or a series of interrelated programs. A software system implies a group of interrelated programs. New software development is generally a standardized process through which software evolves from an idea to a useful system operating on a computer. The traditional model for a software development project includes the following phases which may have to take place, to one degree or another:

1. Feasibility study,
2. Requirements analysis,
3. System design,
4. Program design,
5. Coding,
6. Testing,
7. Documentation, and
8. Implementation.

Five of the above development phases are used in this research: (1) Feasibility study, (2) Requirements analysis, (3) Program design, (4) Test and (5) Documentation. Investigation was limited to these phases because they are the most commonly used and standardized development phases available from the data source. Standard definitions for the development phases and the resource consuming activities accomplished within the phases are contained in Appendix 1 and 2. These definitions have been extracted from the user manual for the

software development project control system at the U.S. Air Force Data Systems Design Center [148].

Schwartz [129] reduces the above list of eight phases to the following four major phases:

1. Requirements specification,
2. System design,
3. Programming (coding), and
4. Checkout.

Within each of the listed phases a multitude of resource consuming activities can be defined. These activities represent work to be done, and project progress as the activities are completed.

System development has become a structured professional discipline. Predefined activities have been established, tested and altered. These form a basis for an orderly, continuing process under which system development is carried out by an interdisciplinary project team. These teams include participation by users, system analysts, programmers, computer operations personnel, operating management, and others [131]. Teichroew has even experimented with automating this system building process [145]. Although, relatively well defined, the software development model is rarely straight-forward. The process frequently involves numerous iterations among the phases of development and the activities within the phases. These iterations are a result of the knowledge gained as the system is being generated.

There is general agreement regarding the necessary steps, in any software development effort, whether large or small, regardless of whose list is used. But, Schwartz points out, understanding and agreement regarding the process almost seem to stop there. The only other area of agreement, that he identifies, is that the development of software is a terrific problem, little understood and fouled up with terrifying frequency [129].

Research Objective

The objective of this research is to quantitatively analyze software development resource consuming activities. The most elementary variable will be the difference (both arithmetic and percent difference) between the hours estimated to complete a standard activity and the hours actually expended. Thousands of these standard activities have been estimated and accomplished across hundreds of software development projects at the U.S. Air Force Data System Design Center. The project control system used by that government agency collected these data (i.e. planned estimates and actual expenditures) as they were generated, in contrast to information collected by questionnaire after a project is completed. Using various statistical techniques, the research will deal with only what these limited data can reveal.

In terms of a general hypothesis this research effort can be described as follows:

Hypothesis: That specific resource consuming activities can be identified which consistently have the most significant influence on inaccurate software development project estimates.

Other questions which may be addressed include:

1. How was the development effort distributed by phase--as compared to the development phase estimates?
2. Which activities and phases consume the majority of resources?
3. Which activities and phases have the largest variance between the estimated and actual time consumed?
4. Does the fluctuation between estimates and expenditures of specific activities consistently have some correlation to the known, final project estimate and actual expenditure totals?

Fleischer [54] believes that programming is becoming the dominant cost factor. Management is thus being pressed to closely study the factors that influence cost and productivity of software development projects. The insights to be gained, according to Fleischer, will save money, time and perhaps even the projects themselves.

Research Constraints

The phase of software Implementation and the subsequent software maintenance and modification efforts are not considered in this dissertation. Implementation and maintenance, however, are certainly not insignificant resource consuming efforts. For example, the International Business Machine Corporation (IBM), Systems Development Division, found that, after two years in the field, 76% of the total cost for a particular software release was spent on maintenance as opposed to the 24% spent on development [26]. A Department of Defense study showed that software for airborne computers cost up to $4000 per instruction over the lifetime of the system [141]. However, research associated with estimating the cost of maintenance is closely related to the problem of software reliability and is not within the purview of this dissertation. Furthermore, the Implementation phase has also been excluded because there is some question regarding the reliability of the data in the data source used for this research. Sherman acknowledged the importance of system Implementation and maintenance as follows: "The completion of system testing and the release of the programs for live running is, to paraphrase a famous speech, 'Not the end of the end, but the end of the beginning'" [134].

This is not another study to evaluate programmer productivity. It is suggested that it is the productivity of analysts and managers which should be examined, if in fact productivity per se is a subject worthy of the amount of investigation given to it. The variability of definition and results regarding productivity are addressed more thoroughly in Chapter III, Literature Survey. Judith

Clapp [30] writes that evidence has shown that writing code is only about 15 to 20% of the total cost of software development [15]. Why then the preoccupation with programmer productivity?

The research will not examine project cost or span time directly. In the data base being used the total duration (span time) of the project is not simply related to the accumulation of the hours planned to be, or actually expended, on each activity. Personnel in the Air Force Data Systems Design Center were,more often than not, assigned to tasks in several projects. For example, an activity estimated to consume eight hours might span a week or more. Furthermore, the system which collected the estimates and expenditures did not accurately keep track of an original, overall estimated date for project completion. It is therefore not propitious to analyze the impact of estimating accuracy (in development activities) on the extension of the duration of a project, using these data.

Unfortunately, cost is not a data element in the project control system providing the data base. However, from a positive viewpoint,its absence prevented it from obscuring the elementary data elements (hours estimated and expended) which this research intended to analyze. Adding burden rates, overhead, indirect costs, etc., and converting hours to dollars through some personnel cost table can tend to obfuscate the elementary act of estimating activities.

Research and Data

Formalization of software production implies better definition of the process and the tools, more standardization, and greater control [30, 6]. Dr. Barry Boehm believes that not having formalized and standardized data bases forces managers to rely on intuition when making decisions on software development planning. Software phenomena often tend to be counterintuitive. Given the magnitude of the risks of basing major software decisions on fallible intuition and the opportunities for ensuring more responsive software by providing designers with usage data, it is surprising, says Boehm, how little effort has gone into endeavors to collect and analyze such data. One

of the reasons progress in software project management is slow is that it is just plain difficult to collect good software data. According to Boehm these difficulties included:

1. Deciding which of the thousands of possibilities to measure.
2. Establishing standard definitions for "error," "test phase," etc.
3. Establishing what had been the development performance criteria.
4. Assessing subjective inputs such as "degree of difficulty," "programmer expertise," etc.
5. Assessing the accuracy of post facto data.
6. Reconciling sets of data collected in differently defined categories. [17]

Dr. Boehm emphasizes that more work on these factors is necessary to insure that future software data collection efforts produce at least roughly comparable results. However, although the data collection problem is difficult does not mean it should be avoided.

There are a few quantitative measures available for evaluating software. These include:

1. Cost,
2. Speed,
3. Size (number of instructions),
4. Effort (man-months or hours),
5. Efficiency--dimensionless--a comparison of how much hardware is used compared with minimum necessary, and
6. Development time. [78]

The element of measurement being used in this research is "effort," in terms of a comparison of the estimated and actual values reported on resource consuming activities.

J. Farquhar of RAND has summarized the entire issue of the estimation of software development best in the following observation:

> Perhaps a discussion of possible research should close with a restatement of the importance of software estimation. It seems plausible that this activity lies at the very heart of software development management. Unable to estimate accurately, the manager can know with certainty neither what resources to commit to an effort nor, in retrospect, how well these resources were used.

The lack of a firm foundation for these two judgements can reduce programming management to a random process in that positive control is next to impossible. This situation often results in the budget overruns and schedule slippages that are all too common today. [49]

CHAPTER II

THE PROBLEM IN PERSPECTIVE

Management and Estimation

Adams, in his dissertation on the estimation process, has focused on the manager as the cynosural element of this process.

> The manager has been described as a specialist in the art of decision making. This description is particularly apropos, for nearly all of his duties involve making decisions. The manager first identifies and defines a group of alternatives relevant to the current problem. Uncertainty is present since the consequences of those alternatives lie in the future and must be anticipated, so the probable results of each alternative are estimated. The manager's ability to estimate accurately, in the face of uncertainty, is thus one key factor which determines, at least in part, his skill as a decision maker and his success as a manager. [2]

Software development management will continue to deserve its current poor reputation for cost estimating and schedule effectiveness until such time as a more complete understanding of the development process is achieved. Systems are still built like the Wright Brothers built airplanes--build the whole thing, push it off a cliff, let it crash, and start all over again [106]. Other symptoms of management failure include schedule overruns, poor quality products and failure to meet project objectives [8, 121]. At a National Conference of the Association for Computing Machinery, George Weinwurm cautioned that:

> A new technology such as ours cannot continue to grow exponentially so long as it is dependent on artisans for everything. The need is for formalized procedures and standards that permit reasonably capable and well-trained people, with the benefit of such a methodology, to do what artisans could do before--and do it consistently. [154]

In support of this statement J. Farquhar observed that this need for consistency is nowhere more apparent, or more painfully felt, than in the area of "software estimation." Without some rational methodology for software estimation, the entire programming management activity becomes entirely subjective--too subjective with regard to the expense of the human and material resources involved [49, 76]. Jones [72] concurs but observes that objective and subjective methodologies reference an idealized dichotomy. In practice, no methodology is ever likely to be wholly objective nor subjective.

Throughout this study the words estimator and manager are used interchangeably. Managers are responsible for estimates but they normally do not make them. In Adams' research, however, the estimators were the managers which violated the tenet that the individual responsible for a task should make the estimate [2, 128].

Estimation is the assessment of resources needed for accomplishing planned objectives. Practically, software estimates take the form of a judgement of the amount of human effort, in man-hours, required to produce the software in question. Scheduling is the placement of resource consuming activities, intrinsic to the objectives, into a time frame. The activities and assigned estimates are among the most important factors influencing a schedule [150]. It is not news that the software industry is under criticism for frequent and spectacular failures in providing effective and reliable software within a reasonable cost envelope and time frame [105]. Kosy has reported several software development projects which have overshot their original cost estimates by wide margins. For example, software for the Federal Aviation Agency's Air Traffic Control Center initially estimated at $1.8 million, had cost $19 million as of 1970. Developing a replacement for one of the subsystems in the U.S. Air Force Air Defense Command's 496L system turned out to cost nearly seven times more than the original estimates and even at that price was never fully completed [78]. The problem is not new, however. Jones [71] points out that many experts believe that software

development is not so different from other unstructured engineering tasks. Indeed, the difficulty in making accurate estimates and the difficulty in preparing cost and time projections are the same kinds of problems that engineers had to struggle with in the infancy of their science. The computer industry continues to forge ahead in improvements to hardware, linguistics, file structures and operating system capabilities but has given little attention to ways in which to manage the other essential factor in every automated system--software development. Great attention is given to saving nano-seconds in the computer and too little attention to wasting hundreds of man-years in developing the programs which are essential to take advantage of the hardware speed.

Progress in software development management can only be based on measurable performance. But management must be provided a baseline against which to make decisions. Several appropriate measures exist for gauging hardware performance (i.e. cost/bit, add time, transfer rate, byte capacity, etc.) but software has few standardized meaningful metrics [78]. Ronald Smith of IBM [138] believes that the understanding of software management has not progressed far enough to know exactly what data are required, when they are necessary, and in what form they are needed to enable a more optimal management performance. The key to successful management of a software project is valid information on which intelligent decisions can be made. Grossly inaccurate estimates serve as highly misleading decision making criteria. Smith also emphasized that the importance of quality management cannot be understated, for it is clear that management has a major effect on whether a project will succeed or fail. Failure, of whatever kind, in software systems has proven to be very costly.

Crisis in Cost

Much of the literature [158, 17] addresses itself to cost, which is certainly the most significant symptom of the software development dilemma. While each new processing technique or hardware gadget

keeps the market place thriving and lowers the cost of computing per se--the cost to develop a system is expanding exponentially in the opposite direction.

The "Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980's," or CCIP-85, study primarily showed that for almost all applications, software (as opposed to computer hardware) was the major source of difficult future problems and operational performance penalties. However, it was found to be difficult to convince people outside the software industry, primarily because of the scarcity of solid quantitative data, to demonstrate the impact of software on accurate scheduling and operational performance [17]. Cost always has been and will continue to be the most significant decision making criterion. The rising cost of software is creating a crisis in the industry. Kosy has neatly summarized the historical aspects of this crisis:

> Requirements for software functions have grown, both in kind and scope, at an *exponential rate*. As one index of that growth, the amount of code that IBM has provided as standard supervisor and utility software for its line of computers has increased by a factor of 1000 over the last 15 years, doubling every 1.4 years. Not coincidentally, the decreasing cost and increasing power of computer hardware has permitted more elaborate software aggregates to become feasible and brought larger, more capable systems into common use. In contrast, the methods used to obtain correct and reliable software, to specify and design software systems, to make and validate software modifications, and to secure classified data in software systems have not changed significantly over the same period. Despite the technological improvements that, by our estimate, have increased the productivity of programmers on small projects by nearly a factor of six over the past decade, large-scale state-of-art systems have taken inordinate efforts to produce, and the effort has increased nonlinearly the bigger the project. More often than not, large software systems are delivered late and cost more than was estimated. [78]

At the 1968 NATO Software Engineering Conference, K. Kolence took exception to the word "crisis" because it is a very emotional word.

The basic problem, according to Kolence, is that certain classes of systems are creating demands which are beyond our capabilities, our theories, methods of design, and production. Kolence emphasized, however, that many areas of software development are still relatively straight-forward--i.e. payroll systems, etc. However, D. Ross of MIT responded that it makes no difference if one's arms, legs, brain and digestive tract are all in fine working order if one is at the moment suffering from a heart attack. One is still very much in a crisis [106]. No discussion of the estimation problem can be complete without acknowledgement to the significance of the cost symptom which the industry seems unable to control.

Management Pays the Price

The CCIP-85 study revealed the following, somewhat staggering, comparative information regarding software costs. For the Air Force, the estimated dollars for FY 1972 annual expenditure on software was between $1 billion and $1.5 billion, about three times the annual expenditure on computer hardware, and about 4% to 5% of the total Air Force budget. Similar figures hold elsewhere. The recent World Wide Military Command and Control System (WWMCCS) computer procurement was estimated to involve expenditures up to $100 million for hardware and over $700 million for software. A recent estimate for NASA was an annual expenditure of $100 million for hardware, and $200 million for software--about 6% of the annual NASA budget.

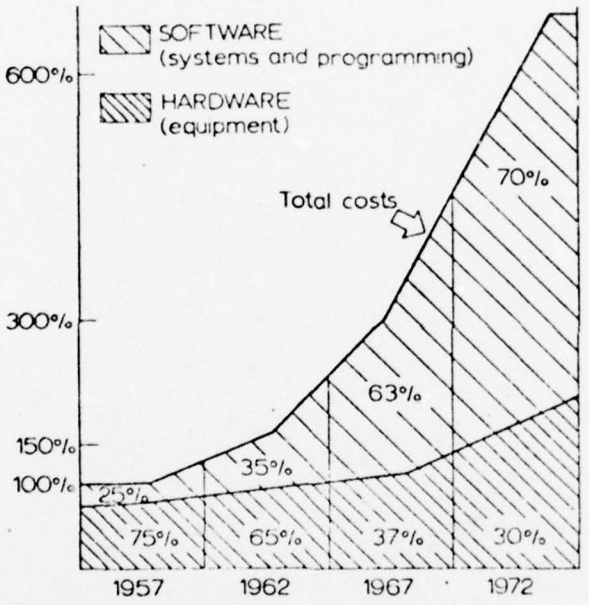
For some individual projects, overall software costs are:

IBM OS/360	$200,000,000,
SAGE	$250,000,000, and
Manned Space Program, 1960-70	$1,000,000,000.

Overall software costs in the U.S. are probably over $10 billion per year, over 1% of the gross national product. Software expenditures in the Air Force are estimated to be going to over 90% of total ADP system costs by 1985; this trend is probably characteristic of other organizations also [17].

If the software-hardware cost ratio appears lopsided now (refer to Figure 1), consider what will happen in the years ahead, as hardware gets cheaper, software systems become larger [125], and costs (people) go higher and higher. Thayer [146] wisely observes that software development is labor intensive.

Fig. 1. Trend comparison of computer system cost distribution in a typical company. [80]



At the 1970 ACM Conference, Dr. Barry Boehm stated that dollar costs for software are in the millions because of project slippages, budget overruns, and what software managers traditionally call "agonizing reappraisal." That is the point where management realizes the software won't work and that large pieces of the system have to be redesigned [15].

Is this cost unimportant? Is it unnecessary that software managers be able to plan, organize, direct and control the efforts of systems development personnel toward efficient realization

of the objective--a system which provides accurate, timely output when it was planned to have it? Is the objective worthwhile at any cost? It is suggested that the answer to these questions is a resounding NO! Sherman also believes that it is not sufficient to accomplish the technical objectives of the project while paying little regard to its overall cost [134]. The industry must begin to look for ways to not only reduce these costs, but to be able to anticipate and control them better. For example, development of Tactical Command and Control Software may require 125 to 3330 man-years and from $7.5 to $200 million, depending on the size and difficulty of the system [78]. The expanded RAND Corporation version of the CCIP-85 study emphasized that budget controls will require minimizing the discrepancy between actual and estimated software costs to 5% to 10% at most. The RAND study also points out that this is not only a problem of cost control but also of arriving at accurate estimates in the first place. While cost estimates rely heavily on human judgement and probably cannot be reduced to mechanical procedures, they should be based, as much as possible, on quantitative relationships and data to minimize subjectivity [78]. But cost is not the underlying problem in software development--management is; and particularly the management function of estimating [154, 25].

The amount of available empirical analysis on estimating computer programming projects is so limited that rules of thumb are substituted for quantitative measures. Speculation is substituted for understanding of the phenomena which influence estimates. It is suspected that herein lies the reason that the basic problem of software development management is continually avoided, and that so little is known of the factors directly bearing on this problem. It seems that in the absence of empirical analysis the industry is limited to working out solutions to cure the symptoms of the problem. The only statement that has been made about system development with absolute certainty is that large systems require more resources and time than small systems. Beyond this truism there is only conjecture [116].

In his book, The Nature of the Physical World, Sir Arthur Eddington observed:

> We often think that when we have completed our study of "one" we know all about "two," because "two" is "one and one"! We forget that we still have to make a study of "and." Secondary physics is the study of "and"--that is to say, of organization. [48]

Organization in Software Development

Two recent management techniques which come closer to bridging the gap between concern for the symptoms and analysis of the underlying problems are both organizational in nature. One technique is IBM's Chief Programmer Teams, and the other is structured programming.

The development of these techniques has been motivated by a desire to reduce the cost of developing and maintaining software by reducing a program's complexity and increasing its clarity. The high cost of programming is due, in large measure, to the complexity of the programs. As a result of this complexity, the program development process is characterized by a large number of mistakes and a great deal of waste and rework. According to Terry Baker there is a persistent myth that programming consists of a little strategic thinking at the top (program design), and a lot of coding at the bottom. But one small statistic is sufficient to explode this misunderstanding. Including all overhead, five to ten debugged instructions are coded per man-day on a large production programming project [10].

The problems of timeliness and reliability are problems of organization as well as technology. To address this, Mills and Baker at IBM developed a programming organization called a Chief Programmer Team [10]. Canning has described a programming management procedure used at Lockheed Corporation prior to 1968 which closely resembles the Chief Programmer Team concept [27].

A Chief Programmer Team represents a new managerial approach to production programming. These changes include restructuring the

work of programming into specialized jobs, defining relationships among specialists, developing new tools to permit these specialists to interface effectively with a developing, visible project; and providing for training and career development of personnel within these specialities. A significant aspect of the Chief Programmer Team approach is to minimize system interactions by concentrating all design decisions on one man. If this aspect is lost, the group rapidly converts into a standard pyramid type organization, structured to control rather than minimize interactions [6].

The second discipline, structured programming, defines a top-down sequence for program unit creation, testing, and a technical standard for the coding of each unit. Structured programming is a manner of organizing and coding programs that makes the programs easily understood and modifiable. Easy modification in turn permits easy maintenance of the product and easy building of a new product using the original product as a base. Structured programming is an evolving discipline which has as its objective more reliable software at less cost and time [29]. Much has been written about structured programming in the last couple of years and its definition varies from writer to writer. However, the fundamental message is "simplify the program control paths" [100, 11].

These are developments that could revolutionize programming in several ways. The most obvious benefits are increased productivity and reduced error rates. An analogy has been made, which the hardware people have known for years, that any logic circuit can be made up from a few basic primitives, such as the "AND" or "OR" operations. Perhaps programming is approaching something of the same maturity.

Analysis of the individual components (Analysis, Programming, etc.) involved in software development may be the best way to understand and optimize the management of each of these phases. Programming itself, as opposed to Analysis or Testing, is probably the easiest and most tangible component to experiment with and study. Nevertheless, research must begin to come to grips with the ineffective planning, estimating and scheduling of the larger and more abstract phases

(Analysis, Design, Testing, etc.) of the software development process. The interrelationships of these phases must also be examined. These enigmatic aspects of the estimating problem will have to be investigated if new projects are to be completed in a reasonable amount of time, and for the total process to evolve as a science.

Reasons for Poor Estimating

The renowned Dr. Frederick Brooks emphasizes that more software projects have gone awry for lack of calendar time than for all other causes combined. Brooks offers five reasons why this cause of disasters is so common.

> First, our techniques of estimating are poorly developed. More seriously, they reflect an unvoiced assumption which is quite untrue, i.e., that all will go well.
>
> Second, our estimating techniques fallaciously confuse effort with progress, hiding the assumption that men and months are interchangeable.
>
> Third, we are uncertain of our estimates. We let the urgency of the patron color (perhaps even govern) our estimate of how long the project will take.
>
> Fourth, schedule progress is poorly monitored. Techniques proven and routine in other engineering disciplines are considered radical innovations in software development.
>
> Fifth, when schedule slippage is recognized, the natural (and traditional) response is to add manpower. Like dowsing a fire with gasoline, this makes matters worse. More fire requires more gasoline, and a regenerative cycle begins which ends in disaster. [22]

Jones believes that historically, the most important reason for poor cost estimates is that the system configuration changes substantially from the time the cost estimate is made until the time the system becomes operational [72]. Conversely, Gildersleeve has found that excessive estimate overruns are less a result of poor estimating than they are of some combination of seventeen (17) various management failures [59, 74].

The stories of resource estimates being missed by factors of two or three are too numerous to be discounted as exaggerated rumor.

Brooks cautions that dangerous reasoning is expressed in the very unit of effort used in estimating and scheduling, the man-month. Cost does indeed vary as the product of the number of men and the number of months. Progress does not! Hence the man-month, as the only unit for measuring the size of a job, is a deceptive myth. It implies that men and months are interchangeable [22].

It is also unrealistic to put managers under pressure to minimize both elapsed time and resources in software development projects. In these instances the manager is not being asked to estimate and schedule, but merely to provide numbers that fit within arbitrarily pre-defined, and probably unrealistic, limits. False scheduling to satisfy a user's, or top management's desired completion date is much more common in the software engineering discipline than in other areas of science. The problem is that it is very difficult for a manager to make a convincing and job-risking defense of an estimate that is derived by no quantitative method, supported by little data, and certified by "rules of thumb" [22].

A New Perspective

At least two approaches are necessary in attacking the problem of accurate software estimation. The first is the more pragmatic: it is necessary to collect data on software project estimates and expenditures as they occur. These data, and the experiences associated with software development projects must be shared. However, even these simple steps would involve a considerable expenditure of money, a serious commitment to the effort, and the courage on the part of management to admit to the errors made during the development cycle. "Egoless management" so to speak, is probably a dichotomy of terms and is too much to hope for. Unfortunately, software managers who must report actual expenditures and failures which occur during development believe that this information will be used against them [30]. Nevertheless, Toffler's philosophy teaches that the imagination is only free when fear of error is temporarily laid aside [54]. Two recent

publications which reflect on the mistakes made during software development are The Mythical Man-Month by Dr. Frederick Brooks [23] and "The MUDD Report" by David Weiss [155]. Boehm suggests it is worth working on a mechanism for transferring government software experience into the commercial sector [15]. The second approach that is needed is for government and industry to support and conduct basic research using the data collected. The problem with these two approaches is: where shall the data come from? Furthermore, estimating accuracy is not the only function of the estimating process [9].

Project Control in Perspective

Mounting expenditures in software development are no longer matched by rising economic returns from that software. Although most of the statements which allude to this state of affairs come from profit seeking organizations, the same general problems are known to exist in other organizations--notably governmental activities [60].

Software projects can be structured and controlled. There are certain activities which must be performed in any software development project. There is also a natural sequence for performing many of these activities. If a software development project is properly defined and structured, management has the techniques and tools available to control the efficient accomplishment of the project [38, 95, 159].

Project control can have some beneficial effect on insuring that software project costs, project duration, and (perhaps surprisingly) project satisfaction, are performed near the limits specified in the project plan. The purpose of project control is to make it easier for management to identify errors or potential errors, and to take action to remedy them. A project will not succeed because of project control but it is unlikely to succeed without it [132, 87, 14, 93]. Control is a state of mind that recognizes a commitment to achieve certain objectives. Control is also the result of that commitment. A manager makes assumptions (estimates), then the control system status

reports compare these assumptions with how the environment is interacting with the assumptions [21, 99]. The problem central to the control and reporting of status is the nature of progress [133, 157]. It is clear that with software development progress the final product cannot be simply described as the sum of the individually completed activities presently used to define the process. Nevertheless, in large development organizations the collection and analysis of project control data should also have a long range benefit in terms of factual experience accumulated. Unfortunately, an organization having a superior system for project control but without the other ingredients of a good project management system is guilty of suboptimization of a most serious kind.

A project control system is, after all, just one major link in the chain associated with managing software development projects. It is also important, to obtain measurable evidence (i.e. data system specifications) to establish that activities were completed [39]. Such evidence also enhances quality control since these products can then be evaluated. This implies some type of broad project control policy.

It would be misleading to suggest that project control is the major problem in the software management area. According to some experts project control is not even a major problem. As evidence, consider the following.

In September 1969, the Founding Conference of the Society for Management Information Systems was held at the University of Minnesota [42]. Attendees at the conference were asked by the University's Management Information Systems Research Center to complete a questionnaire concerning several topics. Most relevant is a question concerning the factors in an organization's environment which are associated with software project success or failure. The respondents rated the overall importance to project success of thirty four factors. Table 1 shows how those factors related to project control, compare relative to other factors.

Table 1. Rankings of Project Control Factors Vis-a-Vis Project Success [42]

Factor	Rank (out of 34)
Measurable project objectives	2
Use of a formalized and regular reporting structure on project progress	13
Planning and accounting for all resources throughout project development	15
Utilization of a formal time-scheduling technique such as PERT for project control	21

A second question asked for a ranking of the importance and priority of twenty six possible research projects in the software development area. Table 2 shows how research, related in some way to project control, was evaluated.

Table 2. Rankings of Potential Software Development Research Projects [42]

Project	Rank (out of 26)
Determination of the factors contributing to the success or failure of EDP projects	6
Development of improved methods for time and cost estimating of systems design projects	8
Development of improved methods for EDP project control	12

Thus, overall, the research project dealing with project control was rated about the middle (12 out of 26). Research to develop better ways of establishing project time and cost targets was rated more highly than research directed toward the development of better project control systems (8 as opposed to 12). Finally, the respondents felt that it

would be better to direct research toward general factors associated with project success and failure than toward project control, per se. One relationship which suggests itself is that it is the lack of adequate management information systems management that is a very great force retarding our progress toward the solution of the problem of collecting and analyzing data and improving estimating accuracy.

For many years, software development projects and data processing managers have been allowed to exist in an environment in which there has been no way of measuring their performance. Data processing managers have therefore not been held accountable for management actions. In other words, software development has been operating out of control for a considerable time. One way to correct this situation is to require the software manager to develop software projects within a control system similar to those applied to other functional managers [143]. Once the process of software development is measurable and visible, it follows that managers and estimators, as a matter of self-interest, will adopt better managerial and estimating practices. Standardized estimating techniques will evolve and data will be collected and analyzed about the factors which influence estimating and decision making relative to the software development process. Project control is significant, therefore, in that it is the best technique for obtaining the measurable data that are required for research.

CHAPTER III

LITERATURE SURVEY

An Old Problem

Since the early 1960's planning and controlling the cost and duration of software development projects have been the foremost management problems in Computing Science. Estimating resource requirements is the one activity accomplished in planning which is least understood and least controlled.

The following paragraphs are excerpts from the introduction at the first symposium on the evolving problems of managing the development of computer program systems. Though written thirteen years ago, it would seem that these remarks are still particularly appropriate to the situation existing today.

> In our concern with the day-to-day management problems of system development, we sometimes forget how young the software industry is. Little more than ten years ago, one man could design, code, and test a typical computer program. Since that time, however, programming has increased vastly in scope and complexity. In order to develop many of today's large-scale computer applications, it is necessary to segment computing problems and arrive at the desired solutions through the concentrated efforts of many designers and programmers.
>
> As the dimensions of computer programming have increased, the normal activities of management have correspondingly become more difficult. The software development manager at the present time operates in an environment of changing requirements, limited time, too few qualified personnel, and often, untried equipment. Most of a manager's energy is thus spent in an effort to get the job done.
>
> To our knowledge no previous meeting has addressed itself specifically to the management process in the development of large computer programming systems. There are, we think, two reasons why there has not been more concern about software management. First our heritage stems from the areas of engineering and scientific computation where computer programs are prepared on a relatively small scale, and where management problems do not approach in breadth or depth those that must be faced in developing large computer

> program systems. Second, we have failed to consider critically our management practices. We seem to have been so concerned with getting the job done that we seem to have spent far too little time in planning and developing good management techniques. It is our conviction that we can improve our management methods if we are willing to invest the necessary time. One of the symposium participants expressed the urgency of this notion very well when he said that we must provide ourselves with an information-handling capability no less adequate than the systems we build for other people. [105]

The symposium participants saw nothing abstract in the difficulties of producing a large program system. To them the complex realities of slipped schedules and the chronic shortage of experienced programmers posed concrete problems requiring hardheaded solutions. This pragmatic outlook, which was evident in all of the symposium discussions, was neatly summed up by an unidentified speaker who pointed out that software development managers are faced with a condition, not a theory. In large part, the condition is the result of the immaturity of the software industry. However, other factors also contribute to this condition. Among these are changing customer requirements, accelerated delivery schedules, and the interface problems between the hardware and software components of a system. Other speakers expressed similar views. For example, if managers could exercise better control over their management environment, many development problems would not be so acute. Furthermore, greater environmental stability would help to improve the reliability of estimates and forecasts, and the quality of software production.

The symposium participants, on the other hand, made it clear that the condition is also the product of shortcomings in management itself. They agreed that to a large extent development managers themselves are to blame for the severity of many problems. The chief criticisms stated by the participants were that development managers do not construct efficient management controls, and that they are apparently unwilling to devote the time and resources required for long-range planning [35]. The programming profession traditionally has been deficient in establishing planning factors for the control of

the development process. Managers often grossly underestimate the size and complexity of a system and then find themselves faced with serious production delays and heavy cost overruns.

Nelson [110] believes that basic principles of management exist which are applicable to all forms of coordinative activity. Management may be defined as the process of accomplishing objectives by establishing an environment favorable to performance by people operating in organized groups. The essence of managing is the attainment of coordination or harmony of individual effort toward the achievement of group goals. The management process may be said to consist of the performance of specific functions, namely: planning, organizing, staffing and assembling resources, directing, and controlling. These functions also apply to the management of computer programming projects.

One universal management principle, for example, has been called the "principle of the primary of planning" [77]. That is, planning is the primary requisite to the other managerial functions of organizing, staffing, directing and controlling. This means that the degree of control over a programming project can be no greater than the extent to which adequate plans have been made for the project. It can be less, of course, since contingencies can force modifications in even the best laid plans; but the extent of planning sets the degree of control that is possible. Furthermore, Nelson suspects that inadequate planning is the primary reason for loss of control on many computer programming projects. It is not the comparative newness of the computer programming process, difficulties with programmers, nor technical factors--it is simply that programming projects are not adequately planned in the first place [110, 111].

New Research

In 1974 the MITRE Corporation was contracted by the U.S. Air Force to research the problem of the engineering of quality software

systems. Among the considerations in this research completed in January 1975, were:

1. Effects of management philosophy on software production, and
2. Measurement of the complexity of computer software [31].

In the introduction to this study, Judith Clapp reasoned that the category of quality analysis is just as essential to quality software engineering as to the general area of design and implementation. Without quantitative measures of software the goal of quality software is futile. The collection and analysis of data about software development (underline added) are as critical as development of the software itself. Data are essential ingredients in any engineering activity, but this fact has been neglected with respect to software. Ms. Clapp also emphasized that the conduct of experiments in software production is a worthy subject for research itself, and that other methods for evaluating new techniques appear to be necessary. One area requiring further research concerns methods for collecting and analyzing data about both software development costs, and failures and errors during software development [30].

It is very difficult to estimate the time and cost necessary to produce a computer software package. This difficulty stems from a poor understanding of the production process, the number of disparate factors affecting program complexity, and the limited ability of managers to assimilate and effectively weigh these factors [49]. Conforti [33], has listed what he believes are barriers for the development of simplified project estimation techniques.

1. Managers themselves. Managers realize that systems work is somewhat subjective, and feel they cannot develop an "accurate" method of easily (underline added) estimating projects. There is an overconcern for accuracy.
2. Upper management's preoccupation with back-up material. Operating in a petty mode where every man-hour must be explained requires a detailed, complex process for developing estimates.

3. The programmer himself. He views his job as a highly creative task, as opposed to making an objective examination of a job with the same basic components used over and over again.

Unfortunately, Conforti minimizes the impact of his accurate observations above by stating that since general time standards are known for the basic components of any program, the task of project estimation is quite easy. There appears to be an inconsistency in this comment. While estimating standard components of a program may be relatively easy--estimating a software development project is very difficult and complex. Studies concerning software estimations have identified at least ninety factors that affect the overall cost of just program development [51]. Jackson states that it is difficult to estimate how long it will take to write a program and believes that as yet there is not any satisfactory way of making these estimates [70].

A MITRE study [30] concluded that prior to the start of software development, management must decide what time and resources will be allocated [34]. Resources include money, people and computer time. Elapsed time is another important resource parameter identified in much of the literature [137]. Reliable estimating is an important management function and is necessary to reduce the uncertainty inherent in software development plans and to support evaluation of performance of software projects.

In 1974, Morin [101] at the University of North Carolina, researched the Estimation of Resources for Computer Programming Projects. She described the study as an effort to draw together what is known about the development of accurate or reliable techniques for estimating manpower, computer resources and elapsed time for a computer programming project. This work was primarily a survey and evaluation of the literature on resource estimation for computer programming projects. Working under Dr. Frederick Brooks, a recognized expert on software development [23], the researcher classified the literature into three convenient categories:

1. Studies describing estimating methods,
2. Papers describing estimating guidelines, and
3. Literature discussing the estimating problem.

The first category consists of the most relevant information and previous research, including some specific techniques. The second category discusses the problem and offers suggestions from experience and proposes general rules of thumb. The last category deals with general discussions of programming management, with estimation discussed as one of the many facets of the problem. The research of the literature in this dissertation will adhere closely to Morin's taxonomy of the literature.

Estimating Methods

Estimating methods, depending upon the type of data available, can be further classified into several techniques. Nelson [112] provides four categories, based on the primary calculations required to develop the estimate. These categories are:

1. Specific Analogy. Using known costs of similar projects to determine estimates for resources of a proposed system.
2. Unit Price. Multiplying a previously determined cost per unit for a given resource by the number of units to be delivered in a proposed project.
3. Percent of Other Item. Setting the cost of a part of a proposed project as a predetermined fraction of another part.
4. Parametric Equations. Using an equation that represents the cost of a proposed project as a function of various characteristics of resources expected to be used under the anticipated working conditions.

Lecht [84] has described a representative and workable Specific Analogy estimating method. This method almost entirely depends upon the use of cost data from similar projects to make estimates concerning the cost of the proposed system. Furthermore, this technique

uses past experience of the estimator, but is generally restricted to the scope and size of projects previously produced. Where resources are extrapolated for larger projects, a linear relationship is generally assumed between program size and costs. However, there is considerable evidence which contradicts the assumption of linearity [158, 101]. Other Specific Analogy methods have been described by Krauss [79] and Hurtado [68]. An interesting on-line "conversational" visual display technique for estimating the cost to manufacture a product has been described by Kelly [75]. This experimental approach could have application to software development estimation by standardizing and simplifying the process, and by having the experience data readily accessible [91].

Estimation of cost by government cost analysts has traditionally relied heavily on the costs, actual or estimated, of prior, contemporary or projected analogous systems. Guidance for using experts as a data source for cost estimating is not well documented in published literature according to Jones. Cost analysts rely almost exclusively on their intuitive judgements in tapping such expertise [73].

Estimating methods which could be classified under Specific Analogy include some of the automated project management systems. These systems provide the manager with some assistance in estimating, planning and scheduling. Project management systems also provide a data base of a project's progress and costs as compared to the effort which had been planned. These type systems include the U.S. Air Force's Planning and Resource Management Information System (PARMIS) [148], System Development Corporation's "Software Factory" [20], and many others [149, 102, 7, 92].

Myers [104] suggests that this "experience" method is frequently unreliable due to the following factors:

1. The relationship between cost and system size (number of program statements) is not linear.
2. Projects with similar names (i.e. payroll system) are often very dissimilar in terms of development costs.

3. Manipulations by management to avoid overruns make historical data questionable.

Aron [5] has described a Quantitative Method which is an example of the Unit Price Technique. This method uses the previously determined rate of programmer productivity (deliverable instructions per unit time) to calculate manpower requirements for large system development efforts. Aron confesses that this estimating technique is not a precise method and in fact is not as good as an estimate based on "sound" experience.

Project managers at IBM have developed a technique which can be classified as a Percent of Other Item method [89]. This method determines the Net Program Development Time in man days, using program complexity, programmer experience, and ability. For example, complexity is assigned a certain value according to input/output characteristics and processing requirements (such as calculations and editing). Other phases of system development are determined as percentages of the Net Program Development Time. This technique has the same disadvantage as the Quantitative Method, in that both rely on the estimators capacity to assign weights to nebulous variables.

Attempts to obtain Parametric Equations for estimating resources for software development projects have been unsuccessful. In fact, there are three studies (SDC [50], PRC [61], Eudy [56]) which encompass what can be described as basic research into this problem.

The most significant study was conducted by the System Development Corporation (SDC) for the U.S. Air Force from 1964 to 1966. Interestingly, Dr. Barry Boehm, an acknowledged expert on the impact of software, recently identified the SDC work as the most exhaustive quantitative analysis done to date on factors influencing software development [17]. If Dr. Boehm is correct then there hasn't been any significant basic research in this critical area of software development estimation for more than ten years. The SDC study was performed in three cycles, each consisting of the collection and

analysis of new data for improving the results achieved from previous cycles. For each cycle, data were collected using an after-the-fact questionnaire, completed by project managers on programming projects which had already terminated. Multiple regression was applied to the numerical data used in the SDC studies in order to derive estimating equations. Nelson [110] cautiously observes, however, that although multiple regression reveals important parameters, it does not necessarily represent natural laws. The data collection on programming projects during all three cycles was restricted to costs incurred during the programming phase only, i.e. program design, code, and program test. In 1967, the final product of the three year SDC study was published as a Management Handbook [108] on cost estimation [120, 119]. Although this document is often cited, there are no reported uses of the resultant equations by programming managers. For example, the thirteen parameter equation for project size had a mean of 40 man-months and a standard deviation of 62 man-months [123]. Nevertheless, the study has made a valuable contribution to the understanding of why computer programming resource estimation is so difficult. At least eleven major publications evolved from the work at SDC [50, 51, 152, 153, 82, 107, 55, 108, 81, 32, 109].

Larger and more expensive programming projects require disproportionately more resources than smaller projects. The SDC study observed this nonlinearity during each cycle of the research project. However, even though the nonlinearity was apparent, SDC attempted to fit the cost data to a linear model by deleting some points, while transforming other data points which represented the larger development efforts. Thus, the SDC work rationalized away this important cost relationship. Pietrasanta reported on some investigations, which revealed that as systems increase in size there appears to be a disproportionate increase required in man-hours for development. Perhaps the simplest explanation comes from the definition of a software system. Such definitions prescribe that software systems consist of two elements: programs and interfaces. Pietrasanta's explanation for this nonlinear increase is

that a linear increase in programs ($y = x_i$) is accompanied by an exponential increase ($y = x_i^N$) in the potential interfaces between the programs [116].

An outgrowth of the SDC work was a study conducted by C. W. Eudy and described by John Gayle [56]. Eudy hoped to increase the reliability of the estimating equations by selecting a set of data with more similar characteristics. For example, he developed a set of equations for estimating resources required by programs written in a single language (e.g. COBOL), and for a single computer (e.g. the IBM 360/40). Eudy's equation for manpower contained a dominant factor which indicated that an increase in distance between the programmer and the computer could reduce the amount of manpower required. Gayle suggests that the reason for this effect is that the minor inconvenience of distance motivates programmers to check their work more thoroughly before submitting a job. This reasoning is not supported by any data and it seems highly improbable that distance should be the dominant factor in the equation for manpower.

Simultaneously with the SDC work, the Planning Research Corporation (PRC) [61, 62] also received a U.S. Air Force contract to study the applications, effectiveness and problems of U.S. Air Force information processing systems. In Phase I of this effort, the PRC researchers summarized and structured the Air Force's data processing experience. They believed that the key to retrieving experience information was "workload," which is defined to be quantities of information processed. This variable was chosen because:

1. "workload" was believed to be a causal factor of cost and development time,
2. "workload" could be quantitatively measured, and
3. "workload" factors would be available in a proposal if a thorough analysis of the problem had been performed.

Forty numerical "workload" descriptors were identified. Phase II was to confirm what, if any, relationships existed among the "workload" descriptors, cost, and development time. The PRC study developed estimating equations for the entire development process (encompassing System Design, Programming, Testing and Implementation phases) and subsequent maintenance efforts. Historical data on past Air Force programming projects were tabulated on scales. A plastic template was used to find "relevant" past projects. The relevance of these projects was summed over a set of factors in order to find a group of "most relevant" past projects. Estimates were obtained by referencing data from these "most relevant" projects. Other methods were developed to check the reasonableness of the estimated values of certain factors. The statistical methods used in the research were: (1) scatter plot analysis, (2) correlation analysis, (3) analysis of variance and co-variance, (4) multiple regression analysis, and (5) factor analysis. The PRC researchers themselves noted that the estimating equations displayed wide prediction intervals because of their small sample size. PRC recommended increasing the sample to 200 projects in Phase III. Their proposal for Phase III was not approved by the Air Force and the research terminated.

Estimating Guidelines

Software development is a complex process not thoroughly understood even by those who do it. As a result, the accuracy of estimates depends on the skill of the project manager to assess the influence of many different factors. Until the relative influence of these factors on the development process can be established conclusively, estimates of cost will probably continue to depend on the experience of the estimator, and qualitative rules of thumb which may be manual or automated.

The following four broad rules of thumb are typical of those used throughout the industry for scheduling software development life cycles.

1. Aron [5] adopted a scheduling rule of 30% planning, 20% coding, and 50% testing, from the background work which had been done by several researchers at IBM.
2. Brooks [23] considered 33% planning, 17% coding and 50% testing appropriate, based on several years of personal experience at IBM.
3. SDC [101] arrived at 34.5% planning, 17% coding and 47.5% testing, by averaging the experience data of the time consumed by programmers in four projects developed for the military.
4. Wolverton [158] reports that various researchers have discovered, by empirical methods, that analysis and design account for 40%, coding and debugging account for 20%, and checkout and test account for 40% [36].

In 1972, Boehm [17] reported software efforts expended on large development projects in slightly different categories. In Table 3, the percentage of effort for each development phase by project is given.

Table 3. Percentage of Effort by Development Phase

	Analysis and Design	Coding and Auditing	Checkout and Test
SAGE	39%	14%	47%
NTDS	30	20	50
GEMINI	36	17	47
SATURN V	32	24	44
OS/360	33	17	50
TRW Survey	46	20	34

Probably the most comprehensive tutorial on estimating the cost of software development is a paper by Ray Wolverton [158]. Wolverton reviews the development process and the traditional methods used in estimating, and identifies many of the pitfalls in price estimation. One of the most significant observations is that, regardless of how accurate the original estimates, predictions cannot be fulfilled unless some mechanism for management control is solved in advance. Estimating and project control are usually thought of as two separate functions, but, in reality, are symbiotic. However, as late as 1970, the response to a survey of some 150 management information system specialists indicated that the presence of "measurable objects" was of great importance to overall project success and that other attributes of project control (such as the use of a formal control system) was relatively unimportant [42].

Software is basically an intangible product during most of the development process. Contrary to experience, the assumption is that changes to programs not yet operational are easily made. Since software system modules are not visibly connected, in contrast to hardware systems, the impact of a change is often not readily apparent even to the designers of the system.

Wolverton proposes five management principles (which apply throughout the software development cycle) to reduce the problem of control to a manageable size.

1. Develop software documentation (i.e. system specifications) throughout the project which can be used as an instrument by which management controls the project.
2. Conduct technical reviews of system design so that agreed upon baselines are established.
3. Control software physical media (tapes, decks, etc.) to assure a known configuration throughout testing and implementation.
4. Apply stringent confirmation management controls to assure that necessary changes, and their impact on the schedule and the system, are fully understood.
5. Provide a control system and status reporting repository to assure that changes to plans, status, and system configuration, are available to all project personnel and users. [158]

Another important project management activity involves conducting system status reviews so that problems can be identified and update planning can occur as required. Plans and estimates must be kept realistic.

Wolverton describes in detail the TRW methodology which has been used for several years with reportedly good results. TRW's estimation algorithm assumes that costs vary proportionally with the number of instructions. For each identified routine the procedure combines an estimate of the number of object instructions, category (control routine, input/output, etc.), relative degree of difficulty, and historical data (in dollars per instruction) from a cost data base. After these variables are estimated and classified, the various development phases are identified. An estimate is made of the fraction of the total effort to be allocated for each phase. The next step involves defining the activities in each development phase by means of a 25 x 7 matrix, with twenty five activities for each of the seven phases. An associated cost matrix is also introduced. The final step is to provide the schedule data based on the customer's statement of work and other management considerations. Overhead burden rates are also input. The outputs from the algorithm include, (a) cost per routine, (b) a graphic display of the schedule, (c) cost breakdown by development phase activity, and (d) manloading and other details of cost and computer loading. The outputs are considered only as an initial estimate by the cost estimation group, which examines the information in conjunction with other sources of data and the project objectives. The final approved set of estimates is input to an official pricing computer run.

Wolverton is quick to point out that valid data, based on proven experience, are required for any cost estimation process. He emphasizes that there is "no universal model" for estimating software costs accurately.

An earlier effort to assist programming managers in planning the entire software development process was a Planning Guide [52]

developed by SDC in 1965 for the Office of Naval Research. The Planning Guide divided the program development cycle into eight phases: (1) systems analysis, (2) design, (3) program development, (4) coding, (5) checkout, (6) documentation, (7) user training, and (8) turnover. Various cost factors were listed for each development phase.

Pietrasanta [116] divides manpower into two categories: (1) development programmers, and (2) technical support personnel. Development programmers design, code and test. Pietrasanta believes that more estimation errors are made in considerations concerning support personnel than programmers. The ratio of programmers to support personnel varies with the development project, but most projects operate on a one to one ratio. In some projects this ratio increases to a one to two ratio, and in a few large systems, approaches a one to three ratio. He believes that as the size of the system increases, the increase in technical support personnel is greater than the increase in the number of analysts and programmers.

The following statements reflect the variety of productivity and cost estimating guidelines found throughout the literature. Each is subject to a variety of qualifications. Delaney [40] proposes (as a guideline for estimating acceptable programmer productivity) a broad average of ten machine instructions per man-day or an input rate of 3000 to 3600 machine instructions per man-year. Corbató [23] reported in 1968 a mean productivity of 1200 debugged PL/1 statements per man-year on the large Multics system. Ercoli [106] estimates the cost of debugged instructions from $1 to $20 per instruction. A recent Department of Defense study indicates that software for its airborne computers cost $75 per instruction to develop [141].

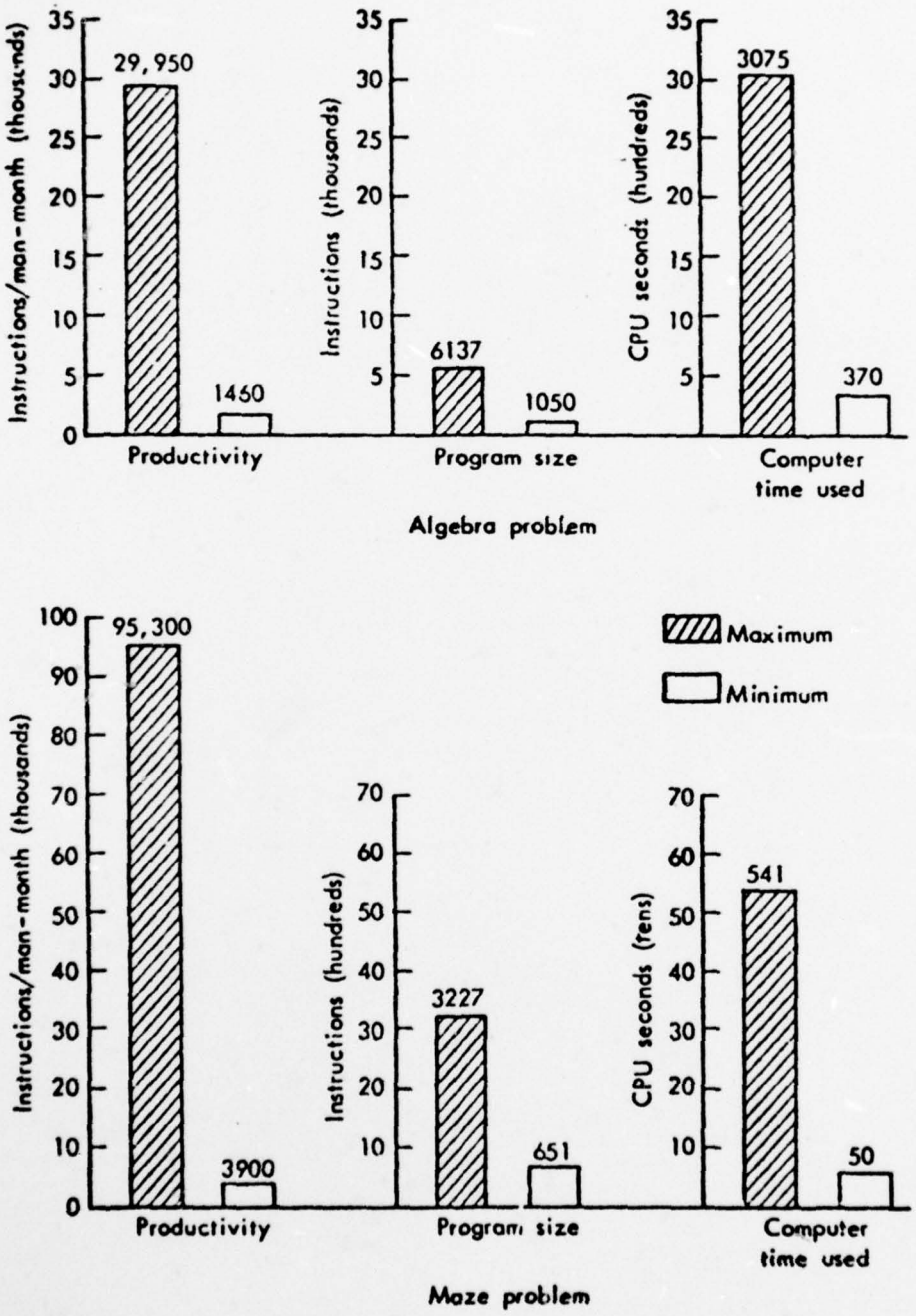
Productivity rates must be used with caution because of the widely varying performance of programmers. At the conference of Software Engineering (Garmish, 1968), David [106] notes an experiment conducted by SDC which reflected a range of 25/1 between the performance of programmers in completing the solution to a specific

logic problem. Much of the difficulty with using productivity rules of thumb in estimating is the confounding effect of a myriad of factors influencing productivity. It is generally accepted today that high-order languages (HOL) and other software tools increase programmer productivity but there is considerable difference of opinion regarding the magnitude of the increase. Corbató states that the number of checked-out lines of code per day, using either assembly language or a HOL, is approximately the same [78]. It can be reasoned, therefore, that productivity, for a HOL, would increase by a factor of 4-6, equivalent to the average number of machine instructions generated by a compiler per source statement. Data from SDC on 14 HOL written programs (JOVIAL) and 60 assembly-coded programs indicate increases by a factor of 1.7 [82].

SOFTECH Inc. reports the use of a number of the new software management techniques with impressive results. They include structured programming, Chief Programmer Teams, and improved specification methods. SOFTECH has reported in 1974, programming productivity of over 12,000 lines of debugged, documented and delivered PL/1 code per man-year [140].

In an experiment to evaluate the difference between time-sharing and batch programming, Sackman gave the same two problems (called Algebra and Maze) to a group of twelve programmers having an average experience of 6.5 years. He concluded that the extreme difference between the programmers' performance on identical tasks totally submerged any differences that might have been found due to the different production methods [126]. Some of the differences are shown graphically in Figure 2. These differences did not correlate with programmer experience. However, the highest productivity in each task was achieved using a HOL (JOVIAL), and the lowest productivity using machine language. The extremely high productivities also highlight the effect of program size and difficulty (i.e. small, easy programs).

Fig. 2. Individual differences in programmer performance on two small problems [126].



Kosy [78] and Weinberg [151] identify additional factors which influence programmer productivity. It should be clear from the summary of observations and experiments that no standard definition of productivity exists. The numbers presented are not comparable because of the differences in how productivity is measured. Thus, any assessment of trends in productivity, or the use of such a nebulous entity for accurately estimating softward development, is generally an exercise in futility.

The major thesis of an IBM technical report by Myers [104] was the consideration of risk relative to cost estimates for software development. Risk is defined as the probability that the actual costs will exceed the estimated costs which Myers viewed as the inverse of confidence; that is, if the risk is 25%, then one can be 75% confident that the actual costs will not exceed the estimated costs. Myers also presented a software development project estimating technique. The estimate is expressed in the form of a probability distribution function, allowing the following types of questions to be answered:

1. What is the probability that actual costs will not exceed x dollars (or man-months, etc.)?
2. For a risk of x percent, what is the estimated cost?
3. What is the probability that actual costs will be between x and y dollars? [104]

Myers limited the use of the technique to projects requiring fifteen or more people and the duration of the project to a range of eighteen to thirty months. The technique requires estimates of input variables such as the number of program instructions, program difficulty, program size, and average competence of the staff. Each variable is given three values (pessimistic, most likely, and optimistic). Difficulty and size variables are multiplied together to obtain a complexity index. Using this index, productivity values (instructions/day), for either assembly language or source language programming, are extracted from tables contained in the automated system. Ultimately three values of the man-months required are determined by simply dividing the estimated number

of instructions by the productivity values. To these computed values are added estimates for "support" manpower using Pietrasanta's [115] findings (i.e. depending on the size of the project, support requires from one half to three times as much of the direct technical manpower needed). The complete estimate, including the associated risk, is developed using the beta function commonly associated with PERT for describing the time necessary to perform a function. Albeit a method, there are a considerable number of assumptions made which make this technique as unpredictable in terms of estimating accuracy as most other methods. At the outset, Myers states that it is relatively easy to estimate the expected size (number of program instructions) of each module, if (underline added) a modular design of the system has already been accomplished. However, requirements analysis and system design are major portions of the software development process and cannot be ignored in the total project estimating endeavor. Pietrasanta [115] points out that the validity of the estimate decreases as the list of assumptions grows in making estimates.

In contrast to Myers' computation of risk, Aron [5] prefers the following discussion of "safety factors." Safety factor is defined as the probability that the actual costs will not exceed the estimated costs. It represents the estimator's confidence in understanding the problem and the likelihood that the assumptions will be satisfied. Safety factors are converted into additional costs and time to allow for errors in the estimate. Given an estimate which includes a safety factor, the project manager may take a "risk" as proposed by Myers. The risk represents the manager's belief that the estimate can be achieved in spite of its potential error.

Considering the record for grossly underestimating software development and the heightened awareness of the complexity of this task, single valued estimates for software development should be considered inadequate for proper management. Conventional estimating

typically introduces the most likely value for each element or activity to determine the total economic implications for the system to be developed. Since distributions of cost are known to be skewed upwards, a single value estimate computed this way has very little likelihood of occurring and may constitute a serious understatement of the true cost. Where a single value is provided, the mean or the modal value is generally more meaningful. In systems development efforts, which are characterized by considerable research and development, these latter values (mean, modal) have been shown to exceed the conventional estimate by more than 30% [1].

It is apparent that a conventional estimate is no better than the quality of its inputs. An estimator's ability to identify all cost contributing factors, combined with the necessity to assess each known factor's economic implication, bears directly on the validity of the end product. It is just as difficult to appraise the range of uncertainty which influences each cost factor. Nevertheless, the manager must diligently search for, and determine the magnitude of, factors which give rise to estimate variability. It is true that the estimator's ability to describe uncertainty is subjective. However, because of the general unavailability of completely relevant historical data from which estimates could be more accurately gauged, single valued estimates also contain varying degrees of subjectivity. It is significant that, although estimators in the past have been aware of these uncertainties, there did not exist a means to assess the cumulative impact on the total estimate. Techniques for evaluating uncertainty are available and should be used [114, 139, 44, 64]. Murphy [103] has described an analytical method which does not improve the precision of an estimate, but which does place the estimated cost value in perspective with respect to decisions made using uncertain values. Although these requirements are an added burden, the estimator will undoubtedly derive greater satisfaction from pursuing this task than

trying to generate a single number in the face of imperfect information. Even the simple appoaches proposed by Myers and Aron provide some indication to management of the capriciousness of software development estimates.

A checklist offered by McClure [106] consists of statements to be considered prior to undertaking new software projects. For example, one statement notes that, "Interfacing this system to the rest of the software is trivial and can be easily worked out later." For each statement checked on the list McClure recommends adding 10% to the estimated cost and one month to the estimated time.

Schwartz has neatly summarized the various maxims used in planning and budgeting software development efforts. Some of these are listed below. He points out that facts available to assist in estimation are limited in preciseness but they are based on considerable experience.

1. It is generally accepted that software costs are more than hardware costs.
2. The dollar cost per ultimately delivered computer instruction is one measure. Estimates which vary from $1 to more than $50 an instruction are not unusual with $5 for average "fair" small jobs. Larger and more complicated jobs lead to higher potential costs.
3. Underestimation by a factor of 2.5 or more is not unusual. Overestimation is almost unheard of.
4. For tasks similar to ones performed previously by the same people, estimates within 10% to 30% are not uncommon.
5. Instructions/hour/man for some large systems can be around 3, or go as low as less than 1. For 100,000 instructions, this may mean at best 10-50 or more man-years; for 1 million maybe at best 100-500 average man-years or more. Estimates of much more than this have been made for some efforts such as IBM's OS/360, where some estimates state that 0.2 instructions/hour were realized.
6. All estimates are functions of complexity, people newness, tools, size, hardware, programming language used, and other factors.
7. Other quantitative values for the programming process have been given. One is that of the total time (after Requirements Analysis) 30% is devoted

to design, 40% to programming, and 30% to system testing. These are approximately equal to other estimates. (Programming includes initial checkout in this case.)

8. Management and support use half of the resources. In other words, for every programmer assume one other person, not a programmer, on the project.
9. During the last half or more of the project, computer time averages around 8 hours per month per man.
10. Averages are dangerous. To live by averages in this case is like saying the average temperature in a desert is 60° (0° for half the year, 120° for the rest).
11. The rate of spending increases over time.
12. There is a frequent desire to shorten elapsed time by various methods, such as adding more people. But one can compress schedules only so much. Also, paradoxically, estimates have been made that show that the same job done in a shorter time using more people costs more than one where more time is used. [129]

Schwartz concludes that the available qualitative ideas do not provide a precise quantitative estimating capability. There is little quantitative help for deciding the "complexity" of a system, the quality of the requirements, the type of programmers, the quality and timeliness of the tools, etc. However, he suggests that since these ideas are based on experience they counter the natural urge to be overoptimistic and should not be ignored. While these ideas and aids do not represent natural laws, Schwartz believes that they are statistically accurate. The following is an often quoted observation from the writing of Alfred Pietrasanta regarding the qualitative and quantitative aspects of estimating software development.

> The problem of resource estimating of computer program system development is fundamentally qualitative rather than quantitative. We do not understand what has to be estimated well enough to make accurate estimates. Quantitative analyses of resources will augment our qualitative understanding of program system development, but such analyses will never substitute for this understanding. [116]

It is apparent that there is a wide variety of techniques, lore and experience that are used in attempts to provide some kind of guidelines to estimating software development. It is also interesting to note that very little correlation has been found between estimating ability and various levels of educational accomplishments [49]. Lulejian and Associates, under contract to the U.S. Air Force, analyzed several software development characteristics for a package of eighty eight routines developed by TRW Systems Corporation. When visually plotted, with use of regression analysis, the data indicated (a) that larger routines cost more, and (b) a linear fit to the data is not very good. Furthermore, the variables of program difficulty and programmer experience could not be shown to be of any help in estimating [18, 63].

Literature about Estimating and Management

In a doctoral dissertation at Syracuse University, Adams [2] investigated the estimation process to determine what factors influence the accuracy of estimates. His interview data and quantitative analyses both strongly supported the five following hypotheses:

1. A positive correlation between accuracy and amount of information sought and processed.
2. A positive correlation between managerial talent and amount of information sought and processed.
3. A positive correlation between the perceived importance of accurate estimates and the amount of information sought and processed.
4. A positive correlation between the measure of managerial talent and the perceived importance of estimates.
5. Estimates tend to become a target to those who must accomplish the activity.

Adams states that his primary conclusion is that the accuracy of estimates in a project management environment is controlled by the estimator to an extent that even he himeslf is not aware of.

Tsichritzis [147] identified two schools of thought for the difficulties in managing software projects: (1) poor management, and (2) random activity, i.e. the production of software is not a deterministic activity since specifications are modified, personnel change, and planning tools are lacking.

Dr. Frederick Brooks [23] has written an excellent book which can be used as a pocket reference to DO's and DON'T's of software development. The book provides a compendium of the mistakes most software managers have made along with the techniques being tried by many. Brooks suggests that software managers are gutless estimators and eternal optimists. There also seems to exist a fear of updating sacrosanct plans. The emphasis in planning should not be, as hardware engineer P. Fagg suggests, to "take no small slips," to allow the work to be done carefully [23]. Managers should emphasize required rescheduling, as necessary, so that everyone is not working frantically toward an impossible goal which ruins morale, as well as the system, and inevitably fails to meet the deadline. Brooks reflects considerable insight into the nuances of management of software development and the psychology of technicians and first level managers in the reporting of problems and delays. Detailed planning and project control are not yet as generally accepted and applied to the development of software as they are to other "engineering" research and development efforts. Brooks also addresses the "role conflict" in project control systems, i.e. distinguishing between action information and status information. Managers also attempt to use project control systems for personnel evaluation and control. Managers have difficulty distinguishing between project control and people control. Using the data reported by people to manage those same people results in unreliable input and a loss of control. People must be managed by people, not by automated systems. Everyone speaks of the need for better data but there are no real commitments to collecting it.

There can be no question that Dr. Brooks knows whereof he speaks but the book is no more than what the author claims--an

accumulation of the lore in the field and his personal views. If software development management is frustrating then this is a book about frustration which leaves one a little frustrated, for it offers no new thoughts on how to do the job better.

The most recent paper on the subject of estimating was again sponsored by research funds from the U.S. Air Force in cooperation with the U.S. Army [137]. The report is an expensive literature survey which concludes that (1) there is no widely accepted methodology for estimating resource requirements for a software development project, and (2) structured programming technology does not introduce new techniques for estimating. The technical report also draws the following conclusions from the survey of the literature on software estimating.

1. Four resources are critical in the management of software development. They are manpower, computer time, money, and elapsed time.
2. The quantitative techniques being used to estimate manpower resources use formulas and average productivity tables.
3. The similar experience technique of comparing new system requirements with experience gained on similar completed systems is the most frequently used method for estimating resources.
4. Software estimating studies which have sought ways to make programmers more productive have failed to first establish a baseline of productivity.
5. The key to more accurate estimating is the collection, retention, and availability of valid historical data. [137]

This writer participated in a study by Scott [130] on the factors affecting programmer productivity and the cost of software development. Scott used the Delphi procedure which has three essential features: (a) anonymous response, (b) iteration and controlled feedback and (c) statistical group response. These features minimize the biasing effects of dominant individuals, irrelevant communications and group pressure toward conformity. While Delphi is a useful device for producing consensus, Farquhar [49] points out that it is not a magical tool for producing a right answer. The people surveyed were either selected because they were

currently managing programming projects, or because they were considered experts in the management of programming projects. Panel members were asked to correlate, on a scale from minus seven to plus seven, the effect of increasing the magnitude for each of 35 variables on programmer productivity. One of the variables which remained highly controversial as to its importance to programmer productivity was "program size." The study results did clearly demonstrate the importance that programming project managers place on providing the working programmer with a well documented, thoroughly defined, independent task. Hill also prescribed documentation as the control element for software development [53]. Experienced programmers working in high level languages also emerged as important factors.

Weinberg's book, The Psychology of Computer Programming [151] which discusses, among other things, the concept of egoless programming, will have significant influence on the management of software development. Briefly, egoless programming means a programmer does not own his programs. Programs are not an extension of the programmer's self image but merely things the programmer happened to work on. Weinberg studied the psychological factors affecting programmer productivity and concluded that personality is a more important factor than intelligence in programming. That is, there are more people unsuited temperamentally for programming than deficient in ability to learn the job [65]. Fleischer [54] points out that a trained person, communicating with other humans, is rarely an obvious failure. However, a programmer communicating with computers becomes accustomed to the experience of frequent total failures! These experiences can lead a person, who identifies himself with his programs, to have a very defensive state of mind. Dick Brandon [19] has described the programmer's personality as excessively independent to the point of mild paranoia, egocentric, slightly neurotic, bordering on limited schizophrenia. Brandon probably said this with tongue in cheek, for Weinberg claims that many software houses and other programming teams have built their success on an egoless programming organization. If Weinberg's ideas had to be

summed up in one sentence it would be that "the human element of computer usage is important" [54].

To test the influence of assumed goals, Weinberg gave four programmers the same problem. He directed two of them to develop a fully checked out program that was as efficient as possible in computer time. The other two were asked to do the job as quickly as possible. The latter group used on the average, only 40% of the machine time and 33% of the effort of the other group, but these programs ran ten times slower on the average. These observations affect estimating only in the sense that it is important for a manager involved in estimating to take into account the influence of organizational objectives and individual personality.

Myers [104] observed that it is also important to separate the terms of "estimate and goal." Since "work expands to fit the allotted time" (or cost) failure is assured if "goals" and "estimates at x percent risk" are equated. Goals should be represented by smaller values according to Myers. However, the estimating problem of software development has not been the expansion of work to fill allotted time, but a significant over expenditure of resources far beyond the allotted time. Estimates must be made as accurately, intelligently, and honestly as possible if those involved are going to support a plan. That is not to say that external commitment dates for the final product cannot be established sometime beyond the estimate. In this sense Myers has reversed the idea of estimate and goal and is certainly talking about different goals than those which influenced productivity in Weinberg's experiments.

Carter, Gibson and Rademacher [28] conducted a study for the U.S. Air Force Office of Scientific Research in an attempt to isolate the factors which determine the success or failure of a system development effort. The study states that planning, control, and implementation of a system development effort could be accomplished more efficiently if management were able to quantify each critical factor in the system undertaking. Wright stated as

a result of his work in information systems, that there are a limited number of reasons for system failure. As in any other type of analysis it is important to segregate the "vital few from the trivial many."

1. System too sophisticated and ambitious.
2. Application not sound.
3. Systems people assumed--and management abdicated--responsibility for system design.
4. Designed to supplant--not support--the user.
5. Optimistic implementation.
6. Company incapable of managing with a system. [28]

The Carter study reduced the critical factors to fourteen through mailed checklists and followup surveys. Factor clusters were then determined, and five general classifications emerged. The four lowest rated factors were generally unrelated and were perceived to be of little importance (i.e. Employee Resistance, User Attitude Toward Design Team, etc.). The other four classifications were (1) User Involvement, (2) Management and User Attitude, (3) Organizational Planning and Control, (4) Systems Expertise. According to the researchers a technique for measuring organizational planning and control of the systems effort is under development. Carter also reports that the research group has underway a search for tests, scales and other instruments to measure personnel expertise. One of the objectives of the study is to determine factors critical to successful system development. What is "successful" system development, is the question which begs to be asked. If Carter's group finds simple methods for quantifying the generalized classifications of factors noted above, what will the measurement of these classifications tell us in terms of predicting the "success" of a development effort?

Under U.S. Air Force sponsorship in the CCIP-85 projects, Dr. Barry Boehm, head of the Information Sciences and Mathematics Department at the RAND Corporation, suggested several opportunities for reducing software delays. These fall into three main categories:

1. Increasing each individual's software productivity,
2. Improving project organization and management, and
3. Initiating software development earlier in the system development cycle [17].

The concern for improvement of programmer productivity is a concern for the smallest factor affecting software development. Table 3 (page 37) clearly reflects where most of the software effort is expended. It can be seen from the table that only 15% of a typical software effort goes into coding. Clearly, then, there is more potential payoff in improving the efficiency of analysis and validation efforts than in faster coding. Certainly these development phases are more difficult to estimate accurately than is programming. Boehm also reported that the largest source of Defense Department software costs, in both the development and maintenance phases, is the cost of accommodating user requirement changes.

The entire development process must be examined. The time consuming and schedule destroying, cyclic development which occurs during the program and system testing is directly related to incomplete problem definition and analysis. The CCIP-85 study group recommended improving programmer productivity and initiating software development earlier in the development cycle [17, 24]. These accommodations are still attempts to administer to only the symptoms (high cost, long development time) of the delay dilemma created by poor estimating. Dr. Boehm, in his article, did not address the underlying problems of improving total software project organization and management (estimating).

Boehm did acknowledge, however, that the problems of software productivity on medium or large projects are largely problems of management: thorough organization, good contingency planning, thoughtful establishment of measurable project milestones, continuous monitoring on whether the milestones are properly passed, and prompt investigation and corrective action in case they are not. In the software management area, one of the major difficulties is the

transfer of experience from one project to the next. For example, many of the lessons learned as far back as SAGE are often ignored in today's software development. In 1961 Hosier published an article on the value of milestones, test plans, precise interface specifications, concurrent system development, and performance analysis, etc. [17]. If it were known where most of the time is spent during development, then subsequent research could attack these time consumers [12]. Nevertheless, beyond these familiar concessions to classic management theory, the CCIP-85 study group offered the Air Force no new approaches to identifying why it doesn't work for software development. TRW Corporation found that they could improve their estimates by increasing their understanding of exactly what steps and processes were involved in software development, thus enabling better management; the better management, in turn, improved estimates [158].

Another paper worthy of note is "The MUDD Report" [155]. This report is the result of a year-long investigation into Navy software problems. The report chronicles the development of a mythical software system and describes where and how the developers went awry. It is based on more than thirty interviews with people responsible for development of various kinds of Navy software in ten different organizations. The emphasis is on the difficulties which caused schedule slippages and cost overruns. Although fictionalized, the report is an excellent example of the kind of post mortem analyses which should be accomplished for large software efforts that fail to meet their deadlines and far exceed estimated costs. An analytical vehicle is needed to promote sharing of valuable experiences, both good and bad. Judith Clapp [30] states the managers who must supply data on actual expenses and failure rates which occur during development feel this would be used to judge them.

Tom Gilb [57, 58], an EDP consultant in Norway, speaks of an emerging technology in software metrics. He strongly believes that all interesting properties of software are indeed measurable

enough to provide practical control over the cost and effort of these properties in a system [3].

The book Practical Strategies for Developing Large Software Systems [67] is a collection of papers by some of the most renowned researchers and practitioners in managing software development. It is a result of a one week seminar held at the University of Southern California in 1974 on the "Modern Techniques for the Design and Construction of Reliable Software." The intent of the seminar was to collect experts from the industry who were intimately involved with the software productivity issue. By emphasizing the industrial point of view, the attendees obtained a true assessment of the extent to which various new strategies were actually being practiced. The editor, Ellis Horowitz, observes that the industry's inability to effectively construct complex software systems has impeded its ability to make good use of computer resources. Furthermore, Horowitz emphasizes the role of the manager and programmer in system building environments. The problems encountered in choosing the right people, tools, schedules and procedures, will, if done improperly, nullify the effects of the finest technology.

Canning [27] has proposed a method on how to manage computer programs effectively. The management techniques used to make a profit on fixed price software development projects, in various companies, are cited [83]. At the 1975 International Conference on Reliable Software, R. Williams also described some techniques found to be successful in managing the development of software [156].

Almost everyone in the industry could provide a comprehensive, valid list of the reasons why software managers cannot estimate accurately [135, 144]. One obvious reason is that managers refuse to accomplish the work necessary to do the job well. Managers seem to be afraid to try to master this task. Equations are sought which will do the job and remove the manager from any culpability. Estimators seek anonymity by hiding behind general rules of thumb. Methodologies used employ vague factors like "complexity" to hedge

estimates [37]. Myers has listed eight of the major reasons for poor estimates:

1. Programming is still in its infancy and does not approach the level of most primitive engineering disciplines.
2. The nature of a "system" is not well understood.
3. Estimators do not understand the economics of computer programming.
4. Estimators do not fully understand the system development process.
5. Estimators assume that "all will go well."
6. Estimators do not connect risk and cost. Cost is a probablistic, not a deterministic variable.
7. There is a lack of reliable historical data on project costs.
8. Most programmers are optimists. [104]

A New Beginning

Malcom Jones [71] of MIT observes that, "although many unknowns are faced in developing new software systems, management should not use this as an excuse for abdicating its responsibility for planning." This is particularly inexcusable where a good control system is available.

Judith Clapp [30] of MITRE Corporation has focused on the common problems in software development:

1. Reduce the cost of computer software development, acquisition, and maintenance, and
2. Improve the quality of software products.

But more important, Ms. Clapp has put her finger on the approach needed to resolve these problems and the ongoing proliferation and fragmentation of effort.

> The referenced research is necessary; much of it is of excellent quality. Our problem and our objective is to distill, synthesize, exploit, and develop these efforts into a coherent body of engineering knowledge and methodology. [31]

CHAPTER IV

DATA SOURCE

Data Availability

The continual lament of researchers in the area of software planning and estimating has been the unavailability of adequate historical data [49, 110].

Dr. Barry Boehm has stated that one of the major problems encountered in a large study done for the Air Force regarding data automation implications of the 1980's [17], was the dearth of hard data available on software efforts which would allow the researchers to analyze the nature of software problems. Judith Clapp, in another, more recent, study done for the Air Force on Engineering of Quality Software Systems [30] noted that the inability of management to correctly estimate software development can be attributed to:

1. The lack of definition of the work to be done,
2. The lack of reliable historical cost data, and
3. The lack of understanding of major factors which affect time and cost.

The New Data Problems

As software development project control systems become more prevalent more historical data are being accumulated. Data reliability, consistency of meaning, and appropriateness of the data to specific research are concerns which have replaced the original problem of data availability.

Reliability will be predominately affected by the control system and by the management atmosphere within the organization. That is, data collection and reporting procedures in a project control system significantly impact data reliability. How

the information (from the control system) is used by management can enhance or destroy the reliability of the data. For example, inaccurate reporting may result from using a project control system for man-hour accounting as well as for projecting costs and status reporting. False reporting to reflect a productive eight hour day is a defense commonly used by personnel wary of such automated spies [96].

Standardized definitions of the resource consuming activities along with consistent units of measure should be established throughout the industry and the government. If such standards are not forthcoming there will continue to be a serious problem with the consistency of meaning of the historical data among different organizations.

Clearly, the data elements, and their particular meanings, in the various project control systems may only be used to support a specified, limited, set of research objectives. That is, the data must be appropriate to the questions examined. Today's project control systems collect data for project status reporting and management information, as opposed to also accumulating complementary data which could be most useful in research. Such complementary data might include: (a) reason codes for small cost and schedule changes or overruns, (b) information regarding changes to system design and their total impact on cost and schedule, (c) recording of assumptions on which original plans were based, (d) basis for estimates, etc. This is, however, a subject worthy of independent research. Therefore, until some standards are established and some way is found to minimize or distribute the costs of data collection, research is limited to the data which are currently available.

DiCola observes that the basic problems in scheduling creative resources are the lack of reliable data and the lack of understanding of the interrelationships among elements of the data. Reliable data require definition of the tasks that must be performed to complete

a project; time estimates for completing each task; a record of actual man-hours expended for each task; and knowledge of the personnel resources. With this data the actual man-hours expended on a task can be compared with the estimates, and the accumulated data can be analyzed [43].

The data used in this research were extracted from historical tapes produced by a software development project control system which the author managed and helped to design over a period of more than two years. As noted in Chapter I, the scope of this research has been purposely restrained by the known limitations of the data base. This research does not seek to solve the problem of software development estimating but attempts to develop better understanding of that process. Thus the data which have been selected are appropriate to the objectives of this research. By limiting observations to only one set of data from one project control system within a single organization (which has a standardized management environment) the data across various projects will be as consistent as is possible. Finally, regarding the question of data reliability some compromise must be made. The data were extracted from a project control system which was not a man-hour accounting system. Nevertheless, a fear of the control system persisted throughout the organization which influenced the integrity of reporting. Any system which relies on the integrity of individuals reporting on their own productivity will always suffer to some degree with biased, inaccurate data. Equally important to any evaluation of the results of this research is some knowledge about the organization which developed these software systems and the internal project control system which collected and processed the data.

Organizational Environment

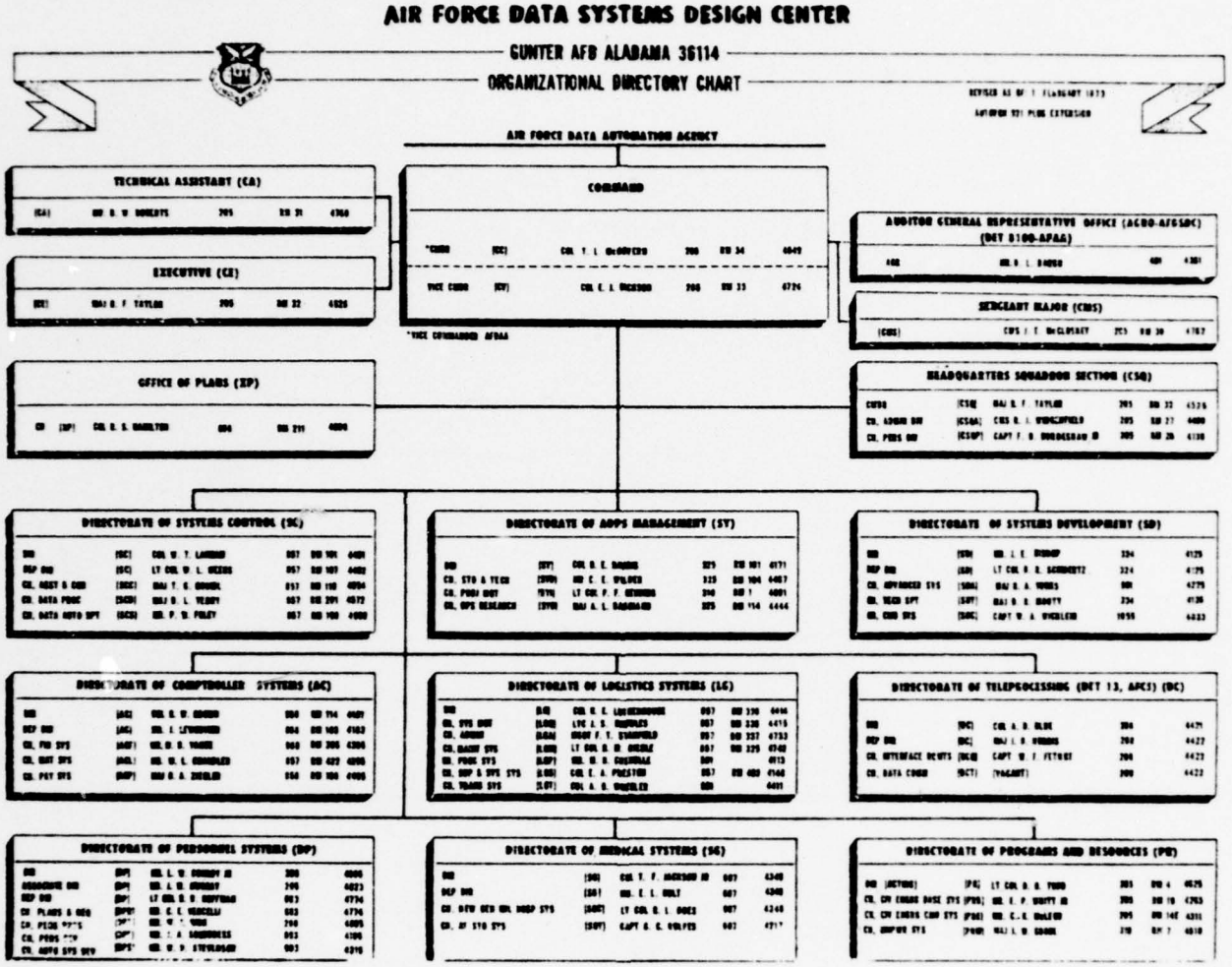
The Air Force Data Systems Design Center (AFDSDC), at Gunter Air Force Station, Montgomery, Alabama, is a unique organization

which is responsible for the total development (Feasibility through Implementation) and continual maintenance of a variety of standard automated management systems used by Air Force installations throughout the world (Figure 3--standard systems are those used at two or more bases). These systems assist the Air Force in managing civilian personnel, aircraft maintenance, hospitals, supply, manpower, etc.; almost every management function in the military service which lends itself to the efficiency and accuracy of data automation. One effect of this variety of functions is that approximately 700 software projects of varying magnitudes are ongoing most of the time. These projects range from a few short weeks for correcting program errors to projects which may involve more than a hundred people over several years.

The Design Center is organized into several functional directorates such as Logistics, Finance and Accounting, and Medical, and supporting data automation directorates, such as Operations and Data Processing Systems Management (Figure 3). The latter organization includes functions such as the development of standards manuals for Air Force automation and a small simulation laboratory. The Operations Directorate runs a closed shop program/system test laboratory. This directorate possesses standard configurations of hardware systems used throughout the Air Force; such as the Burrough's 3500 computer, the Univac 1050 II, the Honeywell 800/200 system and 6500 computer. This organization is also responsible for all software and documentation reproduction and distribution. In addition, the Operations Directorate maintains a unique 24 hour, seven-day-a-week trouble desk which receives calls from around the world regarding software malfunctions.

The Design Center also includes small staffs from the Office of the Auditor General and Air Force Communications. These organizations represent their specialized interests with regard to the design of new management systems to be implemented world wide. Directing the entire professional organization is a commander with his essential staff elements for budget, personnel matters, etc.

Fig. 3. Organizational chart.



In the functional directorates, functional system analysts work side by side with data automation analysts and programmers. More than one thousand personnel are assigned to the Center. Personnel in the Design Center are approximately 60/40 mix of civilians and military respectively. According to a director's management philosophy, the automators and functional analysts might have different supervisors of their own Air Force speciality classification. In contrast, the three types of skills (functional analyst, data system analyst and programmers) could be placed under one supervisor (usually functional) using a project team concept. The supporting data automation directorates are predominately staffed by experienced data automation personnel.

New systems requirements and major modifications to existing systems are assigned by Headquarters U.S. Air Force. The requirements are then staffed by a team of professionals (i.e. the primary functional directorate, interfacing directorates, the Auditor, Communications, Operations, etc.) at the Design Center. This team determines gross requirements for computer support and manpower, and makes a rough estimate of start and completion dates according to the priority of the task. How the proposed system interfaces with other system development efforts and existing systems is also examined.

Project Planning

The approved software development project is then planned in varying degrees of detail. Factors influencing the detail plan are (a) the project manager's understanding of what is required, (b) his experience and management approach, and (3) the specific phase of development being planned. The outline for this standard plan is provided by the Design Center's automated project control system called the Planning and Resource Management Information System (PARMIS) [148]. The Design Center found it imperative to implement some standardized planning and project status sytem with

over seven hundred software development projects ongoing at any one time. Only about sixty of these projects involved more than two man-years of effort (@ 2000 hrs./man-year) but consumed 80% of the available professional resources.

PARMIS was implemented in 1968 as a rather simple but standardized planning and reporting tool. The system was little more than an inventory of ongoing or planned software development projects at its inception. By 1972-1974 (the two year historical data base selected for examination by this research) PARMIS had become well entrenched, and was more sophisticated in terms of planning, tracking project status and highlighting schedule problems. Continuing limitations of the system included incomplete initial planning, a reluctance to update plans, and reporting integrity. Furthermore, the system has never included cost calculations based on the man-hour data available. Complete reporting of all resource expenditures (i.e. implementation, support hours, travel, etc.) on a project is an unrealized objective.

The two year period from 1972-1974 was selected because it contained the planning history of the most completed projects. Furthermore, a major modification to PARMIS prevented original estimates and schedules from being removed. That is, plans and schedules could be changed, such that project status was determined by progress compared to the latest plan, but original estimates and schedules were retained. Thus, this research reflects an accurate comparison between original estimates and the reported expended hours for project totals, and for each detailed, resource consuming, activity which had been planned in each software development project. Nelson emphasizes that if resources can be measured as they are expended, a more accurate cost history of computer programming projects can be compiled for use in future research [110].

Project Size

The projects selected for this research do not represent large system efforts such as the Apollo System, OS/360, or the SABRE

system. Some large system developments do occur at AFDSDC, but only three of any size (> 20,000 hours) are included in this sample. The sample primarily contains small to medium size projects, although many do satisfy some of the criteria for large projects listed by Aron [5], Dennis [41] and McCubbin [46]. Tsichritzis [147] believes that there is a tendency to eliminate "large" software projects by citing that only three experienced persons were used to produce the initial software for the CDC Star computer. A group of three is quite a departure from the 5,000 people used for the development of OS/360. It is unlikely that these facts can be fairly compared, but the difference is so dramatic the comparison is worthy of some attention. The small, highly professional staff used by IBM in their "Chief Programmer Team" concept while developing a large, complex, system for The New York Times is further evidence of this tendency [100].

Control System Features

Malcom Jones of MIT states that perhaps the hardest part of managing software projects is designing the control system. The system is needed in order to verify whether or not the plan is working and whether the project is on schedule. Without a control system it is difficult to detect early signs of trouble so that schedule or specification slippages can be identified and immediately corrected. "Early warning systems," although highly desirable, are not easy to devise for software projects [71, 98].

The Design Center and the PARMIS system possess several unique features which enhance the data and make them more interesting:

1. The variety of functional management systems developed. Some simulation, utility software, and operating system programming is also accomplished.
2. Various sizes of projects.
3. A common, all encompassing, organizational structure provides continuity to the overall management system.

4. There is only a single control system for all users. Standard activity definitions for project planning is required for all software development projects.
5. Mandatory estimating and planning are required for all software development projects.
6. PARMIS is not a man-hour accounting system.
7. Training in the objectives and use of the system is mandatory for all Design Center personnel.
8. A "standard" high order language, COBOL, is predominate for functional software development efforts.
9. Data are collected as the effort is expended.
10. Data are easily accessible.

The limitations of the data caused by the organizational environment and PARMIS itself include:

1. Data reliability caused by:
   (a) The management environment and misuse of the data for purposes other than for what it was intended (i.e. manpower authorization reductions) created a dislike and distrust of the system.
   (b) Reporting integrity and discipline (accuracy and consistency) are strongly influenced by (a).
   (c) The limited and varied experience, planning and estimating ability of project managers.
2. Limited data caused by a failure to plan the early phases of development in detail. Generalized activities are sometimes used to accumulate work effort and disguise project progress.
3. Non standard activities are sometimes included in plans to satisfy different management approaches. While this flexibility is a planning advantage within PARMIS itself, the inconsistency is a disadvantage in research.
4. The data are limited to one organization. The general advantage of management and organizational consistency

is countered by the limitation of not being able to compare a cross section of the industry.

5. Planning complexity within PARMIS was limited during the two years selected. The ability to create extensive planning dependencies through PERT type networking techniques was not available.

PARMIS

PARMIS is similar to most other project control systems and was designed to minimize the effort required in software development, planning and reporting. A new project is initiated by establishing a new record in the system using only thirteen basic administrative data elements (i.e. project manager's name, organization symbol and routing indicator, project number, etc.). These initial data elements must also identify which of the eight major development phases will be necessary for the particular project along with a project complexity code. A project may involve anyone or all of the phases. A problem in an existing system might only necessitate planning a few programming activities. A manager could also specify a particular development phase more than once, which could allow him to devise a separate plan for every program to be developed or modified within the project. The flexibility of the system allows various levels of planning detail to be accomplished for projects of different sizes. For example, a program error which might only require a few hours or days to correct does not require extensive detailed planning. A small new project could also be established and planned in a few hours on a gross activity basis (i.e. Analysis, Programming, Documentation, Testing, etc.).

Once the project is established in the file, PARMIS outputs a Planning Worksheet for each development phase requested that contains all of the standard resource consuming activities, which might be associated with the phase. The worksheet also provides a guide for estimated hours for each activity extracted from a table within

the system, based on the gross project complexity code provided in the initial input. Estimated man-hours for each activity, provided by the individual responsible for planning that phase or activity, always override the man-hour guide provided by the system. The concept of "complexity" is very abstract and poorly defined. The system intends to provide only a rough starting point based on the project manager's intuitive feeling (complexity) for the effort involved. These estimating guidelines based on complexity fall short of being considered an accurate planning tool or aid. In March, 1974, the Design Center Operations Research Division conducted an analysis of the predictive capability of recorded experience. The analysis used the "complexity" code (there are five levels defined) assigned by the planner to each activity as the key input parameter. One finding was that the range of hours recorded in PARMIS history, at each level of complexity, was too broad to be meaningful for predictive purposes [124, 142]. On the Planning Worksheet, the planner handscribes estimates of the time required to do each activity selected. The time is estimated by four categories of resource, namely, functional analyst, data system analyst, programmer and support. Span days for each activity are also estimated. There is not necessarily a relationship between man-hours estimated and the span days required, since most personnel worked on more than one task. If available, other data elements, such as the name of the individual to do the task are also included on the worksheet. Standard and non-standard activities can be added and deleted giving the planner total control of his plan.

The completed plan is input to the system, so that start and completion dates may be derived for each activity within a phase. This computation is based on the start dates provided, a ten year internal calendar, and the simple network relationships established among the activities. In the period 1971-74, network planning of activities could not be accomplished across development phases.

Dependent and independent relationships among activities could only be identified within a phase. The plan is processed between the system and the planner as often as changes continue to be made. When the planner specifies that the initial planning is completed the original plan is firmly established as a project.

Subsequent updated plans and estimates can be submitted as often as necessary, but the original plan remains unchanged after it is accepted as a project. These original planning data are retained for future research, and as a baseline for tracking project progress. Furthermore, this approach should assure a conscientious first effort at planning, and provide planners with an historical look at estimating tendencies.

Data collection and reporting become the primary considerations after the project is started. PARMIS also outputs a unique planning tool which is most beneficial to the individual workers. Every week, each individual scheduled to start to work on, or to complete an activity in any project, receives an Individual Work Schedule List (IWSL). This form lists previously planned work for each individual to be started or completed over the next two week period. The IWSL is also the document on which daily annotations of expended effort, by activity, are made. Each week, after the new IWSL is received, the old form is turned into the PARMIS group for keypunching and entry into the system.

Various management reports trace the status of the projects based on the hours expended by each skill category as compared to hours estimated. Activities completed compared to their forecasted deadline dates are another status indicator. Problem activities and phases are automatically identified.

Another unique output available from PARMIS is an eleven month forecast (limited by the width of the paper, but any eleven months could be selected) of the resources planned to be expended by project, by skill, in any single month. This product can be sorted in a multitude of ways. It can, for example, be used by an organizational entity to determine how their resources are already committed for

each month when planning some new work. By sorting this product by name it shows how an individual is committed across all of the projects he is assigned to. This report frequently helps workers with supervisors who have overcommitted available man-hours in any given month. Clearly, in these cases, some revised planning is essential if all of the work is to be completed on time.

Since PARMIS is not a man-hour accounting system, there is no delay in updating the file with reported progress. The system does not have to insure that everyone reported or that they had reported forty hours. Reporting discipline (reporting every week, on time and accurately) is the responsibility of the individual and the project manager. Data not input on time, failed to be included in standard summary reports for a specific reporting period.

A telephone service is also provided to PARMIS users, for follow-up on system identified deliquent items, for reporting hours, for identifying completed events prior to the IWSL being submitted, and for updating plans and estimates, etc. Each time a plan is changed, the project manager receives a notice of the change and its impact on the schedule. Presently, PARMIS is experimenting with an on-line capability which allows anyone to update the plan directly or to determine immediately the present status of any part of a software development project plan. PARMIS also uses a generalized inquiry routine for special requests for unique management summaries not issued on a periodic basis.

The PARMIS group consists of twelve personnel. Three programmers are used for software maintenance. There are four operations personnel, including one keypunch operator. These personnel assist project managers with planning by helping them to take advantage of unique system capabilities. The Operations section processes all incoming and outgoing products, operates the telephone followup service, and interacts with the computer center for processing. A three man Management Analyst staff prepares periodic briefings and special reports and brings project progress problems to the attention of

project managers. A division chief and secretary complete the organizations authorized manpower. Approximately twenty hours of computer time were used each month (prior to the on-line capacity) for normal processing, special inquiries, and testing of system changes. The Design Center considers PARMIS both an operational necessity and an experimental laboratory. The Center has long range commitments to find a better way of planning, estimating and managing software development.

Data Selection

The data used for this research were selected from the PARMIS history tapes of projects completed during the years 1972-1974. A week of research at the Air Force Data Systems Design Center in Montgomery, Alabama, obtained several different output products from the PARMIS history tapes using a generalized inquiry routine:

1. The first report listed all projects. It reflected which development phases had been planned and the number of activities which had been worked on in that phase. The report also provided phase and project totals for the hours expended by each of the four skills, their sum, and the sum of hours estimated. From this listing, candidate projects were classified as "good" or "possible." The twenty one projects identified as "good," containing 3225 activity records, were all used in the research because they reflected relatively complete development projects. That is, each project usually spanned the gamut of development phases and each phase contained a representative sample of activities. Only eighteen of the twenty one "possible" candidates were ultimately used, representing 850 activity records. These are projects with incomplete planning or some other deficient planning characteristic. However, it must be remembered that the primary objective of the research

was to examine estimating accuracy as it related to specific activities. An unexpected result would be obtained from the research if some correlation could be identified between activity estimates and total project estimates. More than 210 completed projects, available in the data base, were rejected for various reasons including:

(a) The project represented special studies. For example, only the Feasibility phase was included.

(b) The projects only involved some documentation changes.

(c) The project planning was incomplete (i.e. involved programming only). No Analysis, Documentation, Testing or Implementation phases were included.

(d) Too few activities had been planned within the development phases.

(e) Projects were also rejected where it was apparent project managers had manipulated the PARMIS system by assigning different project numbers to each different development phase.

Elimination of a project did not necessarily exclude the activities within the project as candidates for the data base population.

2. The next product derived from the data base was a listing of the detailed activities by assigned identification (ID) number which included the definition of the kind of work being accomplished in the activity. These activities were grouped by development phase within a project. The listing reflected the hours expended by each of the four skills, the total expended, the total estimated, and the numeric difference. It was decided not to include activities where an estimate was made but where zero

hours were expended. It is believed that including these errors "of planning judgement" would unnecessarily bias the examination of estimation accuracy of activities which actually occurred (consumed resources). A minor problem made itself apparent in this listing. In numerous cases the planner had not used the standard activity ID number which rendered useless the special magnetic tape of the two year historical data provided by the AFDSDC. Fortunately, in most cases of this type the standard activity definition had been used and it was easy, though manually time consuming, to convert the ID number back to its standard. There are numerous reasons why planners changed standard ID numbers. However, a discussion of the rationale for such action is not pertinent to this study. Where an activity with a non-standard ID and a non-standard definition could not be clearly related to a standard activity, it was discarded. Project estimated and expended totals, however, were not changed as a result of these kinds of inconsistencies in the data.
Another decision made, as a result of close examination of this product, was to eliminate the Implementation activities from consideration. Attempts to report and accumulate expended hours for Implementation activities, which involved the expenditure of effort from many different organizational entities, was very plebeian at that stage of the development of PARMIS. Therefore the activities from the Implementation Phase are considered too unreliable for use in this research. Again, actual project totals were left intact.

3. The final product obtained was a listing of all activities grouped by activity ID number. This listing also included the activity definition of the work being performed and all summarized hours. The 1200 independent activities (unrelated to specific projects previously selected), which completed the population of records (@ 5250) to be examined in the research, were selected from this listing. The population total represents approximately 59% of the activity records available in the original two year data base.

Data Elements

While the PARMIS data base contains numerous data elements for project and resource control, those which will be of primary concern to this research are as follows:

Project Number: Correlates all activities under one accounting number and identifies the type of application (Finance, Personnel, etc.).

Event Group: Identifies the phase of development (Analysis, Programming, Testing, etc.).

Event Number: Identifies the activity within the phase (coding, typing documentation draft, etc.). The primary element to be analyzed in terms of estimating accuracy.

Event Title: A standard definition of the work being accomplished.

Workload Category: A type of priority scheme. Category A is used for programming maintenance projects (error correction) which will not be considered in this research. This is the highest priority of work at the Design Center. Category B and C include new development projects and modifications to

existing functional systems, utilities and operating system programs. It is from these two priorities that the projects and activities will be selected.

Program Action Code: This code further classifies work under the Workload Category. This research sample will be limited to new development projects (code N).

Estimated Hours: The original estimate of man hours required to complete an activity, subdivided by Functional Analyst, Data Systems Analyst, Programmer, Support and Total.

New Estimated Hours: The latest estimate of time to complete the event, subdivided as above.

Expended Hours: The time required to complete the event, subdivided as above [148].

Throughout most of the research an additional 43 "small" projects were uniquely identified. The "small" projects consisted of 375 individual activity records. These are projects classified under an X35.2 Activity Group in Appendix 1. The activities in this group (Item Numbers 67-72) use only gross definitions for planning (i.e. K100 Feasibility, K200 Analysis, K300 Programming, K400 Test and K500 Documentation). It was decided that these gross activity definitions were incompatible with the standard project activities and should not be included in this research.

CHAPTER V

PROCEDURE

The Elementary Variables

There is no dearth of recommended techniques and experiences for attacking the symptoms of the estimation problem (software development cost and time overruns). These techniques are of value, but do not replace basic research where experience and rules of thumb are supported by measured results. Dr. Boehm concludes that until a firm data base is established, the phrase "Software Engineering" will be a dichotomy of terms and the software component of what is now called computer science will remain far from Lord Kelvin's standard:

> When you can measure what you are speaking about, and express it in numbers, you *know* something about it: but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: it may be the beginning of knowledge but you have scarcely in your thoughts, advanced to the stage of science. [13]

The elementary procedure of this research has been to measure the simple arithmetic and percent differences (d_i) between the estimated (E_i) and observed expended (O_i) hours of standard resource consuming activities (i), accomplished during software development. It is suggested that the arithmetic difference provides a variable which represents magnitude which influences project duration and cost. Percent difference is an indicator of estimating accuracy.

$d_i = E_i - O_i$ a measure of magnitude

$d'_i = (E_i - O_i)/O_i$ a measure of accuracy

Data Elements

Subsequent to the process described in Chapter IV, Data Source, for selecting the data base of projects, the data were keypunched into a record with the following fields:

Phase Code	A single alphabetic code to identify development phase. These codes are the same as used for the six (6) phases in PARMIS. For example: B - Feasibility study D - Programming
Activity Number	Three position numeric identifier which is unique when associated with a specific phase code. Same code as used in PARMIS. Sixty (60) unique activities were used in this study. For example: D020 - coding
Functional Analyst Hours Data System Analyst Hours Programmer Hours Support Hours	Number of hours expended on this activity. Identified by skill designation.
Estimated Hours	Total hours estimated for the activity. Not identified by skill.
Project Number Code	Three position numeric identifier to distinguish among projects and independent activities as follows: 000 - independent activities 001-021 - "good" projects 501-518 - "possible" projects

Computer Programs

The first computer program accepted the above data, edited, and summarized each record both as an individual activity record in the total population (including independent records) and as an activity within a specified project.

The following summaries and computations were made for each unique activity (60 each) and development phase (6 each) for both the

1. population (all activity records), and
2. separate project totals (activity records within a specific project):
   (a) Total expended hours by skill,
   (b) Total expended hours - sum of the four skills,
   (c) Total number of occurrences,
   (d) Number of times underestimated,
   (e) Number of times overestimated,
   (f) Number of times estimated and expended hours exactly equal,
   (g) Total number of hours underestimated,
   (h) Total number of hours overestimated,
   (i) Absolute sum of the arithmetic differences

$$\sum d_i = \sum_{i=1}^{N} \left| (E_i - O_i) \right|,$$

   (j) Sum of the percent differences

$$\sum d_i = \sum_{i=1}^{N} \left| (E_i - O_i)/O_i \right|,$$

   (k) Mean of the arithmetic (and percent) differences

$$\bar{d}_i = \frac{\sum_{i=1}^{N} d_i}{N}, \text{ and } \bar{d}'_i = \frac{\sum_{i=1}^{N} d_i}{N},$$

   (l) Standard deviation of the arithmetic (and percent) differences

$$SD = \frac{\sqrt{\sum_{i=1}^{N} d_i^2 - (\sum_{i=1}^{N} d_i)^2/N}}{N - 1}.$$

In addition to the summary listing, the above routine also provided punched output for each unique activity in each project, for subsequent processing. This output included:

1. Phase code and activity number,
2. The arithmetic difference of each activity record in a project,

3. The percent difference of each activity record in a project, and
4. Project number.

The purpose of a second routine was to "score" (0 or 1) each unique activity, within each project, for subsequent use in SEQUIN (Sequential Item Analysis Routine) [118], based on whether specified criteria were satisfied (1) or not (0). Two criteria were used. One criterion (x) was used to determine if the difference (d, arithmetic or percent), of a specific activity (i), fell within some percentage range of the standard deviation (SD) computed for that activity. Several test values were selected for this criterion (i.e. x = 0.1, 0.25, 1.0, 1.25, 1.5 and 2.0). The relationship between the variable pairs (difference, d_i, and standard deviation, SD_i) is as follows:

$$(1) \quad d_i < x \cdot SD_i$$

The relationship was applied to the arithmetic difference and corresponding standard deviation pair for each activity in each project. The percent difference and its corresponding standard deviation also formed a variable pair against which the relationship was applied. The second criterion was based on whether some percentage (50% or 75%) of the unique (same phase and activity code) activities, within a specific project, satisfied the criterion in expression (1). This latter criterion was specified because most projects had multiple unique activities, while SEQUIN allows only one "score" (0 or 1) per "score card" for each unique activity or phase. For example, activity D020, Coding, would be individually estimated for every separate program in the project. Analysis activities might have multiple occurrences where different personnel worked on separate parts of the activity. Furthermore, this criterion permitted some reasonable flexibility in examining estimating accuracy.

Twenty-four (24) individual "score cards" were output for each project for subsequent input to SEQUIN. This was a result of using the six (6) variable modifiers (x) of the standard deviation in

conjunction with both the arithmetic and percent differences, in addition to the requirement that only 50% or 75% of the unique activity differences satisfy expression (1). A one (1) or a zero (0) was placed in each of 72 columns (one for each of the 60 activities and 6 phases) in each "score card" based on whether the two conditions or criteria (expression (1), and some percentage of activities satisfying that expression) were satisfied. Recall from page 75 that activities 67-72 were excluded from the research.

In particular, a "score" (0 or 1) was obtained by comparing the arithmetic difference (d_i), for each activity in a project to some percentage (criterion x) of the corresponding standard deviation (SD) computed for that unique activity from the total population. Each time a unique activity, within a specific project, satisfied the criterion in expression (1) a counter (k) was increased by one. When all project activities were processed the value for each counter (k) was then divided by the total number of occurrences of each unique activity (M) within the project. If the dividend was equal to or greater than 50% then the appropriate activity column for one of the "score cards" for that project received a one (1). Another "score card," for each project, was output with appropriate "scores" based on whether the result of a counter divided by the total number of occurrences of a unique activity equaled or exceeded 75%. That is,

when $d_{i,k} < x \cdot SD_i$

where i = a unique activity, $i = 1\ (1)\ N$,

k = a dummy index to indicate the number of multiple occurrences of a unique activity within a project, $0 \leq k \leq M$,

M = the number of multiple occurrences of a unique activity in a project,

N = the number of unique activities in a project, $N \leq 60$,

then score = 1

if $\sum_{k=1}^{M} d_{i,k} \geq 0.5 \cdot M$

or $\sum_{k=1}^{M} d_{i,k} \geq 0.75 \cdot M$

otherwise score = 0.

Similar "score cards" were created using the percent difference and its corresponding standard deviation in expression (1). Phases were "scored" by comparing the sum of the arithmetic (and percent) differences (d_i) for all the activities in that phase, of that project, to the corresponding phase standard deviation computed from the total population. In this case,

where J = number of activities in a unique phase in a project

p = a unique phase

then score = 1

if $\sum_{i=1}^{J} d_i < x \cdot SD_p$

otherwise score = 0.

The above algorithms were repeated for each project.

The SEQUIN Parameter Selection Program

SEQUIN uses two types of variables. These are described as either "internal" or "external." Internal variables are values created by SEQUIN from the input "scores" (0 to 1) provided. These "internal scores" are essentially totals (across all projects) of the input scores for each unique activity within each project "score card." External variables are values which are not (necessarily) functions of the input "scores" and which are input as separate data. Examples of external variables used include:

1. Total hours expended on each project (including expended hours on activities within the project which were not used, i.e. non-standard).
2. Total hours estimated for each project (including estimated hours for non-standard activities noted above, but excluding activities with estimated hours and zero expended hours).

3. The absolute arithmetic difference between 1 and 2 above.
4. The percent difference between 1 and 2 above, expressed as an integer.
5. The inverse of the arithmetic difference between the total hours estimated and expended for each project expressed as an integer (all activities were included without exception).

The objective of SEQUIN is to identify parameters which, in general, optimize some function (called the objective function) of validity (correlation), internal consistency reliability, and perhaps other variables. This research is seeking to isolate, through SEQUIN, parameters (activities) which are critical indicators of whether a project has been estimated well. That is, if the estimates for the pool of best predictor activities, selected by SEQUIN, are accurate there should be some confidence that the overall set of activities is estimated well. SEQUIN also attempts to identify a correlation, if one exists, between internal and external variables. For example, SEQUIN has been used successfully, to reduce the number of questions on a test, by identifying those questions which most consistently contributed to discovering what the test was attempting to learn about an individual's knowledge in some area. By using the appropriate SAT scores as external variables SEQUIN was able to find a correlation between the test score, using the reduced set of questions, and the results from the SAT.

The problem of isolating the consistently most influential activities can be attacked by a variety of optimization techniques. One practical way of proceeding, and the method used by SEQUIN, is to use a strategy similar to, but arithmetically less complicated than, the accretion procedure of statistical multiple regression analysis. This procedure is sub-optimal in the sense that it does not necessarily produce optimal solutions. Experience has shown, however, that SEQUIN determines solutions which are significant improvements over those derived by human examination of basic data [118].

Having pointed out that SEQUIN utilizes a searching procedure that is not optimal, but can usually be expected to be practically useful, the specific method is now characterized. SEQUIN first constructs from the parameter responses (activity "scores") based on the criterion used (whether arithmetic or percent differences and the percent of the SD), a basic semi-matrix of the form shown in Figure 4. Then for each project "score card" separately, the program selects parameters (activities) in such a manner as to increase the validity (correlation) of the accumulated information result (internal score) created by the parameter selection process. This selection "evaluation" is the iterative accretion process which will build the list of the most predictive activities using the cumulative validity at each stage of the process. For example, the activity is first selected which had the highest validity (internal score) with the original criteria. Next another parameter is selected such that, when combined with the first selected parameter the two produce "a two parameter list of indicative (or predictive) activities" with the greatest validity. Having found that pair, a new third parameter is sought which, when combined with the previously selected pair, produces a three "parameter set" with maximum validity. This process is repeated until all parameters (activities) are used.

It can easily be shown that for I parameters SEQUIN will investigate

$$(2) \qquad \sum_{i=0}^{i=I-1} C_1^{I-1} = I(I + 1)/2$$

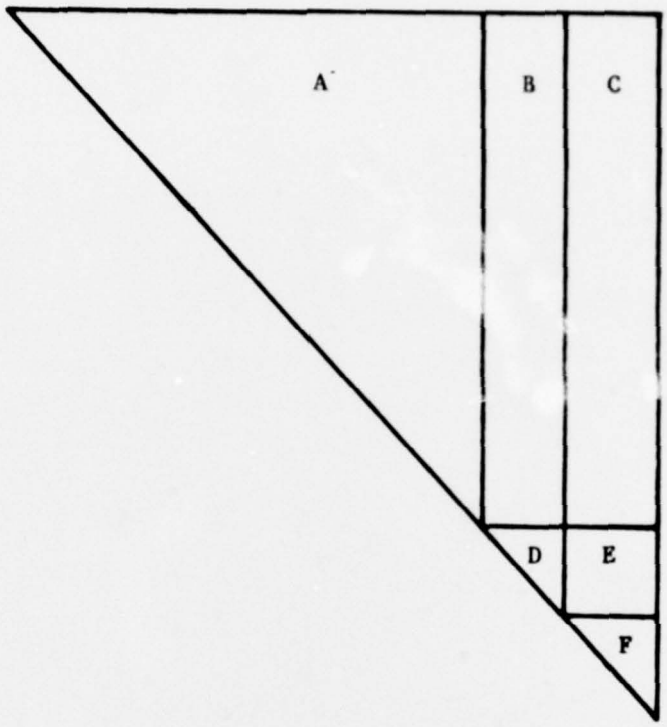
combinations of parameters. Even for I = 30 this requires searching 465 combinations whereas a complete survey requires over a billion combinations to be examined. The price paid for this strategy is the insecurity of not knowing the true optimal in terms of predictive capability of each activity nor how close the results come to the true optima.

AD-A047 674 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO F/G 9/2
A QUANTITATIVE ANALYSIS OF ESTIMATING ACCURACY IN SOFTWARE DEVE--ETC(U)
AUG 76 P F GEHRING
UNCLASSIFIED AFIT-CI-77-28 NL

2 OF 2
AD
A047674

END
DATE FILMED
1 -78
DDC

Fig. 4. Typical information used in a statistical analysis.



SECTION	CONTENT
A	Inter-Parameter Information
B	Parameter Difficulty Information
C	Parameter Validity Information
D	Sample Size
E	Criterion Mean
F	Criterion Variance

The SEQUIN Objective Function

The objective function used in the SEQUIN program is simply a function of the difference of the validity of "evaluations" constructed from k and k-1 parameters (activities). In other words the objective function is expressed as the difference of product-moment correlations of the "evaluations" and the predictor variable (i.e., internal or external variable).

Define the "scored" response (0 or 1) made in the s^{th} project to the i^{th} activity as X(i,s). Then the "evaluation" score (T, or "internal score") of the "score card" of the s^{th} project made up of responses to k parameters can be expressed as:

$$(3) \quad T(s,k) = \sum_{i=1}^{k} X(i,s), \quad s=1(1)S.$$

T is a measure of how often activity estimates for a project were made correctly (i.e., within the relationships established in logical expression (1) and the percent criterion). Designate the predictor variable associated with the s^{th} project as y(s). Also define:

V[k] = variance of the T(s,k),

C[y,k] = covariance of the y(s) and T(s,k), and

V[y] = variance of the y(s).

The objective function, OF[k] can be expressed as the difference between the validity function for k activities and the validity function for k-1 activities. Then OF[k] is written as:

$$(4) \quad OF[k] = \frac{C[y,k]}{\sqrt{V[y] \cdot V[k]}} - \frac{C[y,k-1]}{\sqrt{V[y] \cdot V[k-1]}} .$$

Since V[y] is invariant with respect to the number of activities in the "evaluation" it can be ignored in the computation, i.e. V[y] = 1 in formula (4). Thus,

$$(5) \quad OF[k] = \frac{C[y,k]}{\sqrt{V[k]}} - \frac{C[y,k-1]}{\sqrt{V[k-1]}} .$$

The computational problem is therefore reduced to determining the numerator and denominator of the terms on the right of equation (5).

It is convenient to express C[y,k] as a function of C[y,k-1], and V[k] as a function of V[k-1]. This is easily achieved since,

$$(6) \quad C[y,k] = C[y,k-1] + C[y,i_k],$$

where $C[y,i_k]$ is the covariance of the predictor variable y and the responses (0 or 1) to the parameters (activities) chosen at the k^{th} stage in the sequence (i.e., the k^{th} iteration of the accretion process using "evaluation scores"). Also

$$(7) \quad V[k] = V[k-1] + V[i_k] + 2\sum_{j=1}^{k-1} C(i_k,i_j)$$

where $V[i_k]$ is the variance associated with the responses (activity scores) to the parameter (activity) at stage k and

$$\sum_{j=1}^{k-1} C[i_k,i_j]$$

is the sum of covariance of the responses to the parameter response chosen at stage k with the responses to each of the other parameters in the "evaluation" at stage k-1. The functions are easily computed from information available in sections A and C of the semi-matrix of Figure 4 (page 84).

The fundamental strategy of SEQUIN is to compute OF[k] for each parameter response not in the "evaluation" at stage k-1 using formulas (5), (6) and (7) and find that parameter (activity) for which OF[k] is a maximum. This parameter then becomes part of the "evaluation" (i.e., candidate pool of the most predictive activities) and the next stage is entered. This cycle is repeated until the parameters available are exhausted. During this process, the elements of sections A and C of Figure 4 (page 84) need not be transformed in any way as the computational process proceeds. Thus, although the procedure for seeking the best solution is similar to the accretion procedure of multiple regression analysis, the computations are greatly reduced [118].

SEQUIN Output

The primary output from SEQUIN is a table. A sample computer output from this research is displayed in Appendix 3. The Sequence Number specifies the number and order of activities in the "evaluation" (which make some contribution in predicting overall estimating accuracy on the projects) created by the parameter selection process; those activities making no contribution (i.e., with Item Difficulty of < 0.0005) having already been eliminated by SEQUIN (see Appendix 3, page 173). The Criterion Number specifies the number of the external or internal criterion for which the table is appropriate. The Item Number of the selected activity is specified together with that parameter's Item Difficulty, Point Biserial and Biserial Correlation with the criterion being analyzed from each iteration of the process.

The title "CUM CORR" stands for cumulative correlation which is equivalent to the "validity" between the "evaluation" created by the parameter selection process. The ordering of activities is a function of the parameter selection process which generates this value. This column is frequently the one of greatest interest. The change from the previous iteration of the selection process is given in the column marked "CORR CHANGE."

The Internal Consistency reliability of the "evaluation" is provided. This number is equivalent to that derived from a "Hoyt" or "Kuder-Richardson Formula 20" analysis. The change in the reliability coefficient is also given. The "evaluation" Mean, Standard Deviation and Standard Error of Measurement are provided. Joint Reval is the cumulative correlation multiplied by the internal consistency. Reval Change is the internal consistency reliability divided by the validity change (fr[illegible]urrent and last stage).

CHAPTER VI

FINDINGS

Quantitative Versus Qualitative Analysis

A considerable amount of quantitative information, derived from this research, is displayed in this chapter. While these classifications and analyses provide some valuable insights into the software development process, they should not obfuscate the need for qualitative interpretations as well.

A guiding principle of this research has been to seek only what information the limited data base could provide. Analytical observations should not exceed the known constraints of a data source. The author's considerable knowledge of and experience with, the AFDSDC, the PARMIS system and the data base have restrained any inclination to draw firm general conclusions regarding software development throughout the industry. Hopefully this elementary effort will provoke the beginning of new studies into the improvement of software development management, planning, estimation and control.

This study intends no criticism of the U.S. Air Force Data Systems Design Center management nor of its policies and procedures. On the contrary, the Design Center's commitment to software development project control, the continual experimental improvements to the PARMIS system and the Center's willingness to share the PARMIS data for research, are indicative of the Air Force's concern with the software estimation problem. The lack of software development estimating expertise is not unique to the government.

Quantitative Display of the Findings

The summarized data pertaining to the selected population of activities are contained in Table 4. (Definitions of the activities are contained in Appendix 2.) Totals for the columns titled "Number Underestimated" and "Number Overestimated" were simply computed by

Table 4. Activity and Phase Summary

Activity	Total no. occurrences	No. underest / Hours	No. overest / Hours	No. even	Expended hour totals by skill					Estimated hours total	Arithmetic difference (hours)		Percent difference	
					FA	DA	PG	SP	Total		Mean	Standard Deviation	Mean	Standard Deviation
B100	14	4 / 248	10 / 1146	0	763	516	0	7	1286	2184	99.57	150.78	587.79	1356.20
B110	3	1 / 2	2 / 452	0	18	25	0	7	50	500	151.33	152.07	2351.80	2035.88
B200	78	24 / 1225	52 / 2926	2	5036	2946	263	123	8368	10069	53.22	83.97	387.96	1421.28
B210	4	2 / 137	2 / 330	0	17	157	8	155	337	530	116.75	147.90	113.65	142.09
B300	40	12 / 720	25 / 4189	3	2143	1255	144	224	3766	7235	122.72	164.10	527.83	1765.56
B310	6	3 / 49	2 / 52	1	262	195	0	28	485	488	16.83	18.24	110.91	157.15
B400	65	31 / 1198	32 / 3795	2	2353	2919	52	68	5392	7989	76.82	92.23	301.67	844.82
B410	28	15 / 377	12 / 1147	1	815	1201	0	199	2215	2985	54.43	70.49	128.02	214.62
Total Phase B	238	92 / 3956	137 / 14037	9	11407	9214	467	811	21899	31980	75.60	110.12	382.22	1253.60
C010	56	19 / 713	33 / 2151	4	1275	3546	115	83	5019	6457	51.14	85.83	452.48	1637.05
C020	29	14 / 181	12 / 506	3	1394	556	2	58	2010	2335	23.69	50.89	155.67	370.05
C030	26	11 / 117	13 / 364	2	379	1708	6	56	2149	2396	18.50	31.61	174.17	598.65
C040	41	17 / 301	21 / 829	3	1541	1212	5	26	2784	3312	27.56	45.06	113.03	293.87
C050	98	34 / 1676	59 / 2127	5	2300	3199	246	72	5817	6268	38.81	100.76	291.91	666.06

Table 4. (continued)

Activity	Total no. occurrences	No. underest / Hours	No. overest / Hours	No. even	Expended hour totals by skill					Estimated hours total	Arithmetic difference (hours)		Percent difference	
					FA	DA	PG	SP	Total		Mean	Standard Deviation	Mean	Standard Deviation
C060	27	12 / 399	15 / 380	0	380	2298	113	28	2819	2800	28.85	33.18	126.77	244.96
C070	38	16 / 565	19 / 1144	3	1220	1351	66	63	2700	3279	44.97	78.78	163.08	364.35
C080	2	1 / 296	1 / 35	0	600	327	0	110	1037	776	165.50	184.55	97.90	97.25
C090	21	11 / 696	10 / 454	0	2058	1748	1	169	3976	3734	54.76	89.67	135.31	229.59
C100	16	4 / 190	12 / 374	0	356	851	0	46	1253	1437	35.25	47.67	397.40	630.99
C110	51	23 / 2159	26 / 1347	2	2547	2142	885	151	5725	4913	68.75	113.71	281.22	731.24
C120	115	54 / 2468	58 / 2530	3	4877	4624	178	562	10241	10303	43.46	71.34	200.43	551.08
C130	55	16 / 1851	38 / 3544	1	1063	9753	1683	283	12782	14475	98.09	133.60	111.70	205.33
C140	20	4 / 75	15 / 959	1	31	755	58	6	850	1734	51.70	147.12	202.76	468.63
C150	202	84 / 3373	107 / 2899	11	1933	5456	1603	42	9034	8560	31.05	53.26	188.24	376.88
C160	45	15 / 809	26 / 1708	4	1733	481	616	14	2844	3743	55.93	122.76	264.37	597.98
C161	55	24 / 1259	29 / 1286	2	1190	2235	95	198	3718	3745	46.27	56.69	292.77	693.69
Total Phase C	397	359 / 17128	494 / 22637	44	24877	42242	5672	1967	74758	80267	44.33	84.40	223.24	638.18
D010	405	152 / 7277	233 / 15319	20	3474	5592	16278	83	25427	33469	55.79	135.47	299.28	914.72

Table 4. (continued)

Activity	Total no. occurrences	No. underest / Hours	No. overest / Hours	No. even	Expended hour totals by skill					Estimated hours total	Arithmetic difference (hours)		Percent difference	
					FA	DA	PG	SP	Total		Mean	Standard Deviation	Mean	Standard Deviation
D020	492	201 / 9375	268 / 14005	23	694	3711	27980	235	32620	37250	44.52	98.35	284.15	2006.76
D030	165	38 / 433	108 / 2283	19	6	70	1848	498	2422	4272	16.46	18.80	409.54	828.68
D040	362	115 / 1855	223 / 7706	24	113	212	9575	139	10039	15890	26.41	64.85	309.48	818.13
D050	339	122 / 2509	191 / 5873	26	424	1159	9991	71	11645	15009	24.73	31.22	311.85	853.53
D071	132	26 / 500	93 / 1685	13	906	225	654	16	1801	2986	16.55	30.93	292.21	508.17
D090	673	269 / 10526	352 / 19196	52	5144	3328	26516	373	35361	44031	44.16	131.50	278.71	1714.50
D120	154	51 / 1278	96 / 3738	7	796	720	3885	105	5506	7966	32.57	48.06	364.57	1054.05
D130	212	39 / 984	165 / 4189	8	434	1210	1039	329	3012	6217	24.40	57.48	541.03	1091.08
Total Phase D	2934	1013 / 34737	1729 / 73994	192	11991	16227	97766	1849	127833	167090	37.06	96.56	321.50	1351.71
E010	389	184 / 12958	193 / 5967	12	6568	7136	13784	548	28036	21045	48.65	81.93	193.10	385.47
E020	104	57 / 3481	45 / 951	2	724	1825	3421	32	6002	3472	42.62	65.96	149.80	254.79
E050	31	14 / 344	15 / 307	2	280	528	112	52	972	935	21.00	43.51	181.45	374.48
E070	74	29 / 1347	45 / 1479	0	1716	2495	1110	24	5345	5477	38.19	51.18	206.36	494.06
E080	28	12 / 248	16 / 611	0	40	264	199	51	554	917	30.68	42.27	427.65	728.33

Table 4. (continued)

Activity	Total no. occurrences	No. underest / Hours	No. overest / Hours	No. even	Expended hour totals by skill					Estimated hours total	Arithmetic difference (hours)		Percent difference	
					FA	DA	PG	SP	Total		Mean	Standard Deviation	Mean	Standard Deviation
Total Phase E	626	296 / 18378	314 / 9315	16	9328	12248	18626	707	40909	31846	44.24	73.56	197.38	404.83
F500	2	1 / 3	0 / 0	1	0	1	0	3	4	1	1.50	2.12	50.00	70.71
F520	46	20 / 720	25 / 2095	1	244	3092	715	336	4387	5762	61.20	69.16	358.43	1078.70
F530	9	3 / 34	6 / 196	0	0	454	3	266	723	885	25.56	18.05	97.27	126.64
F540	6	0 / 0	4 / 142	2	4	51	2	10	67	209	23.67	36.03	329.30	534.26
F550	6	3 / 283	3 / 49	0	21	212	0	149	382	148	55.33	71.05	210.78	272.40
F570	13	5 / 226	8 / 115	0	45	100	726	16	887	776	26.23	30.35	348.73	680.57
F620	47	21 / 1212	24 / 958	2	287	1304	1313	590	3494	3240	46.17	62.11	118.93	170.74
F630	8	4 / 38	4 / 355	0	0	227	5	61	293	610	49.13	57.71	390.88	578.68
F640	4	0 / 0	2 / 17	2	4	10	0	0	14	31	4.25	5.32	312.50	529.74
F650	11	6 / 280	5 / 200	0	91	293	38	90	512	432	43.64	55.21	570.61	1185.10
F670	6	2 / 27	4 / 47	0	3	85	3	12	103	123	12.33	15.42	348.85	760.43
F720	25	15 / 607	10 / 382	0	62	1784	67	196	2109	1884	39.36	33.53	59.84	50.52
F730	5	1 / 1	4 / 42	0	0	13	0	40	53	94	8.60	10.14	134.43	176.55

Table 4. (continued)

Activity	Total no. occurrences	No. underest / Hours	No. overest / Hours	No. even	Expended hour totals by skill					Estimated hours total	Arithmetic difference (hours)		Percent difference	
					FA	DA	PG	SP	Total		Mean	Standard Deviation	Mean	Standard Deviation
F740	5	1 / 22	4 / 14	0	0	53	0	0	53	45	7.20	8.44	69.63	51.46
F750	10	6 / 161	4 / 17	0	75	129	1	6	211	67	17.80	27.67	133.92	174.03
F770	14	8 / 283	6 / 296	0	0	789	2	150	941	954	41.36	54.07	113.13	134.00
Total Phase F	217	96 / 3897	113 / 4925	8	836	8597	2875	1925	14233	15261	40.65	53.76	225.87	633.17
G120	84	43 / 4217	38 / 11554	3	8748	572	19	507	9846	17183	187.75	905.92	256.08	1052.62
G130	20	8 / 583	12 / 3373	0	1002	39	0	748	1789	4579	197.80	620.36	513.19	1074.69
G140	8	5 / 108	1 / 158	2	111	6	0	14	131	181	33.25	52.93	1042.07	2771.33
G150	12	8 / 674	4 / 1067	0	594	53	3	224	874	1267	145.08	272.08	377.40	945.25
G170	13	5 / 26	8 / 121	0	204	41	7	8	260	355	11.31	9.96	129.44	149.37
Total Phase G	137	69 / 5608	63 / 16273	5	10659	711	29	1501	12900	23565	159.72	751.36	338.12	1161.56
K100	60	29 / 998	23 / 834	8	1217	2557	237	22	4033	3869	30.53	44.12	85.54	184.90
K200	119	39 / 1641	67 / 2146	13	1095	5626	621	111	7453	7958	31.82	62.24	179.15	564.31
K300	95	30 / 2229	59 / 2635	6	59	1167	6639	58	7923	8329	51.20	79.08	135.08	415.17
K400	107	47 / 1688	49 / 1466	11	452	1824	2053	147	4476	4254	29.48	53.76	165.28	317.41

Table 4. (continued)

Activity	Total no. occurrences	No. underest / Hours	No. overest / Hours	No. even	Expended hour totals by skill					Estimated hours total	Arithmetic difference (hours)		Percent difference	
					FA	DA	PG	SP	Total		Mean	Standard Deviation	Mean	Standard Deviation
K500	93	38 / 2073	49 / 2118	6	445	2319	823	1438	5025	5070	45.06	114.16	331.03	987.62
Total Phase K	474	183 / 8629	247 / 9199	44	3268	13493	10373	1776	28910	29480	37.61	75.58	185.13	579.91
Phase														
B	238	92 / 3956	137 / 14037	9	11407	9214	467	811	21899	31980	75.60	110.12	382.22	1253.60
C	897	359 / 17128	494 / 22637	44	24877	42242	5672	1967	74758	80267	44.33	84.40	223.24	638.18
D	2934	1013 / 34737	1729 / 73994	192	11991	16227	97766	1849	127833	167090	37.06	96.56	321.50	1351.71
E	626	296 / 18378	314 / 9315	16	9328	12248	18626	707	40909	31846	44.24	73.56	197.38	404.83
F	217	96 / 3897	113 / 4925	8	836	8597	2875	1925	14233	15261	40.65	53.76	225.87	633.17
G	137	69 / 5608	63 / 16273	5	10659	711	29	1501	12900	23565	159.72	751.36	338.12	1161.56
F & G	354	165 / 9505	176 / 21198	13	11495	9308	2904	3426	27133	38826				
TOTAL	5049				69098	89239	125435	8760	292532	350009				
PERCENT					23.6	30.5	42.9	3.0						

subtracting the hours expended from the hours estimated for each occurrence of the activity. A negative difference, therefore, corresponded to an underestimate. Skill abbreviations are defined as follows:

FA - Functional analyst. A specialist in the various functional career fields (Personnel, Finance, etc.) in the Air Force. Responsible to define user requirements, for test evaluation, and for preparation of user documentation.

DA - Data Systems Analyst. A specified Air Force career field or a senior programmer. Responsible for system analysis, design, testing and documentation.

PG - Programmer. A specified Air Force career field. Responsible for detailed program design, coding and checkout.

SP - Support personnel. This classification of consumed hours is very broad and may represent keypunch, typist, management, computer operator, and a variety of other types of support personnel.

Computations of the mean and standard deviation for the arithmetic difference between the estimate and expenditure, represent the activities tendency to affect project duration. Since the mean was computed using the absolute value of the difference, it cannot be presumed that the duration is always extended. The mean and standard deviation of the percent difference provide a representation of the estimating accuracy (or more precisely the inaccuracy) associated with each activity. Figures 5 through 18 are the comparative (graphical) representations of the data in Table 4.

Information pertinent to the distribution of activities, skills and hours within each project is summarized in Table 5. A comparison is provided between the "Estimated/Expended Totals of Selected Activities" from a project for the population and the "Estimated/Expended Totals of Actual Projects" from the historical data base. This comparison provides some indication of the amount of data which had to be discarded for the various reasons stated in Chapter IV, Data Source. In deference to the many "rules of thumb" regarding

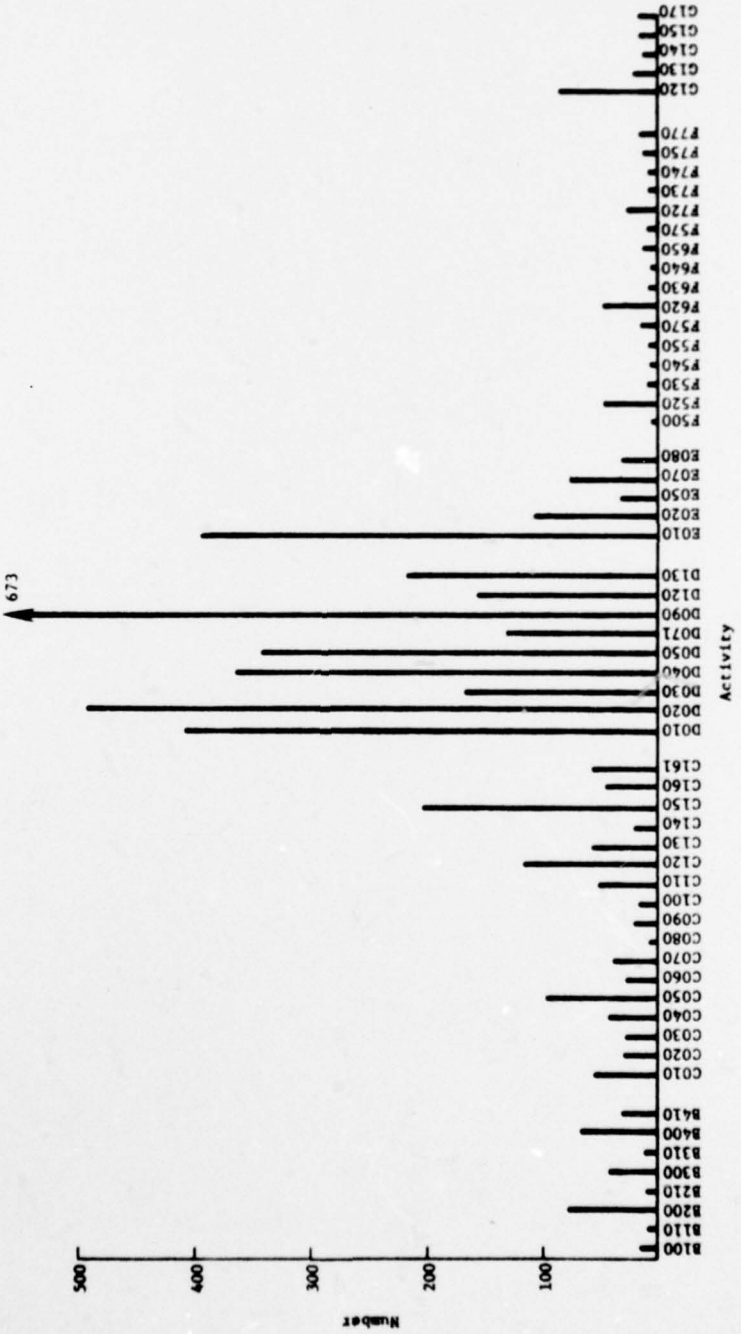
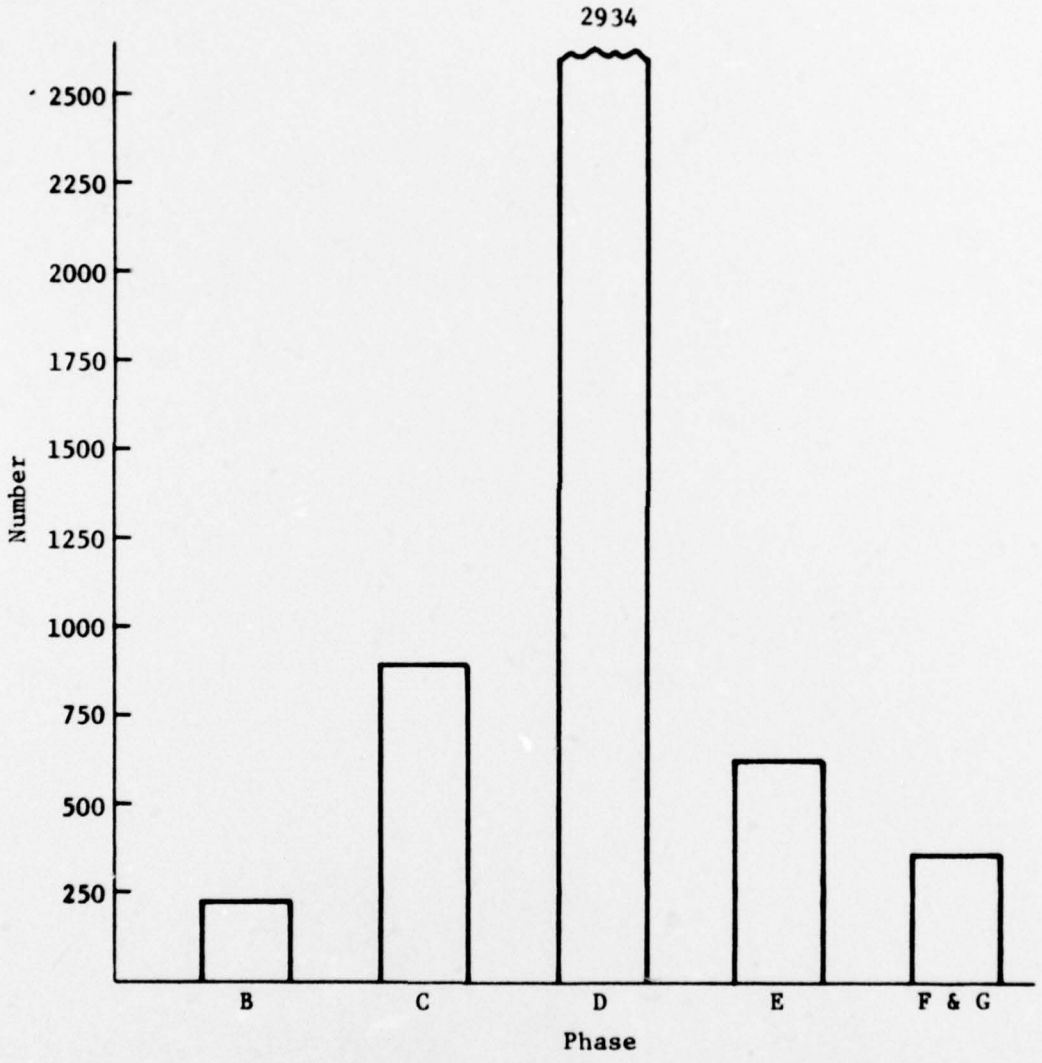


Fig. 5. Number of activities.

Fig. 6. Number of activities/phase.



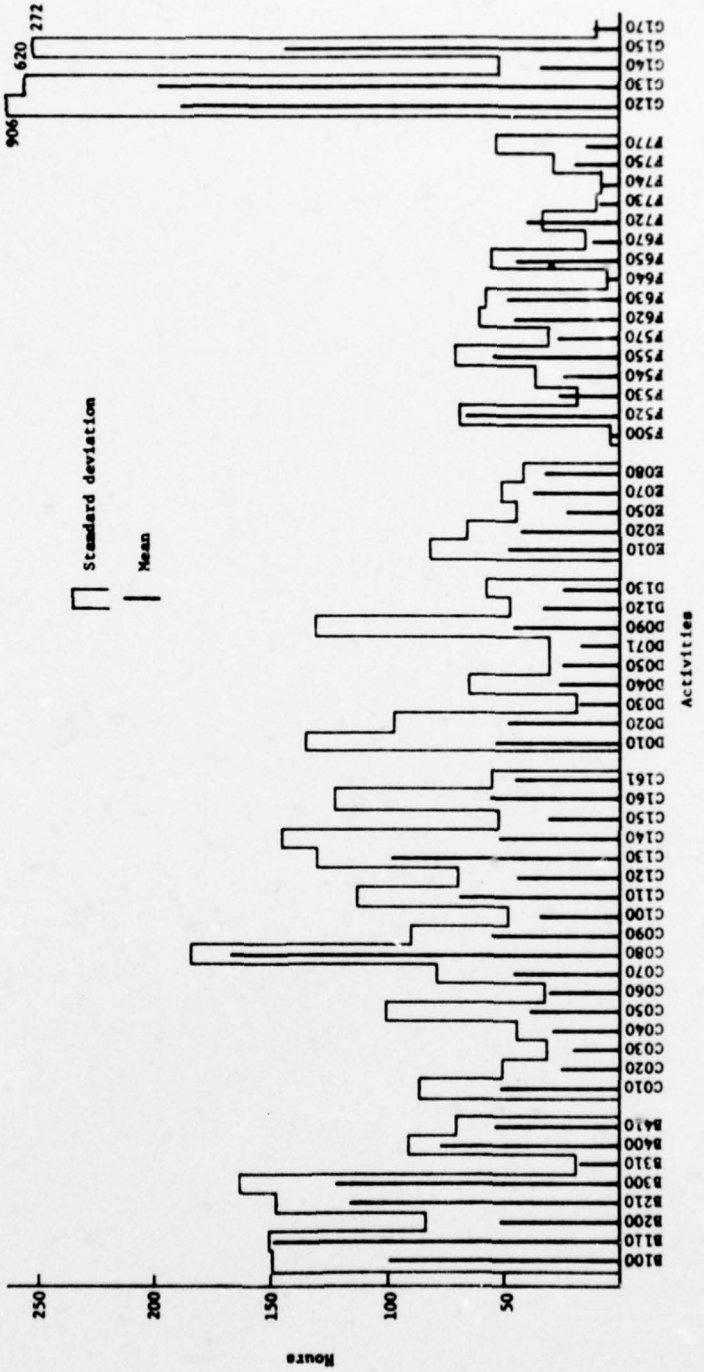
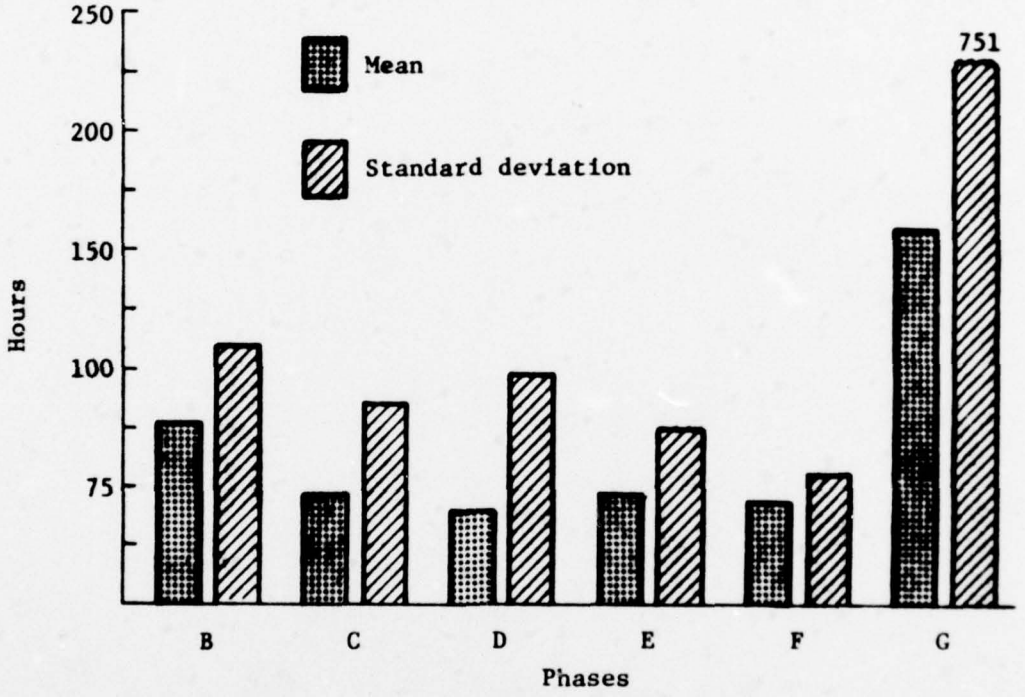


Fig. 7. Mean and standard deviation of arithmetic difference.

Fig. 8. Mean and standard deviation of arithmetic difference/phase.



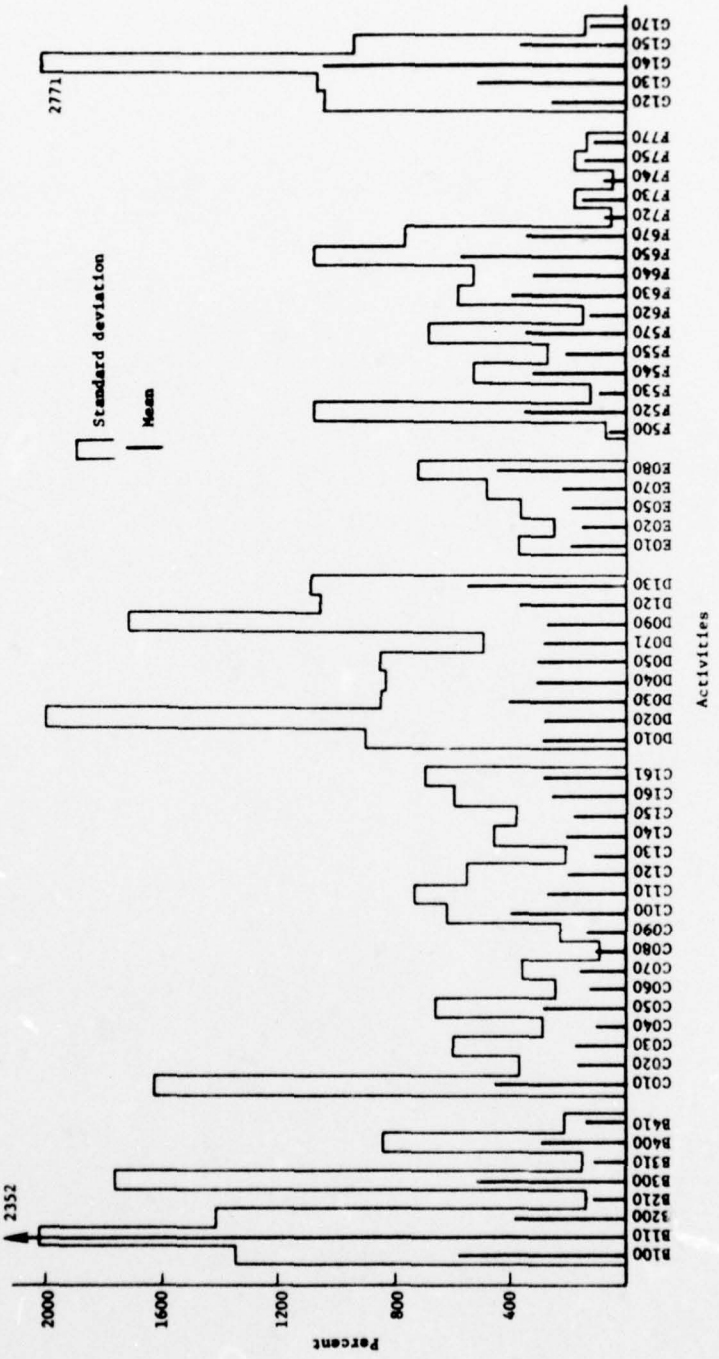
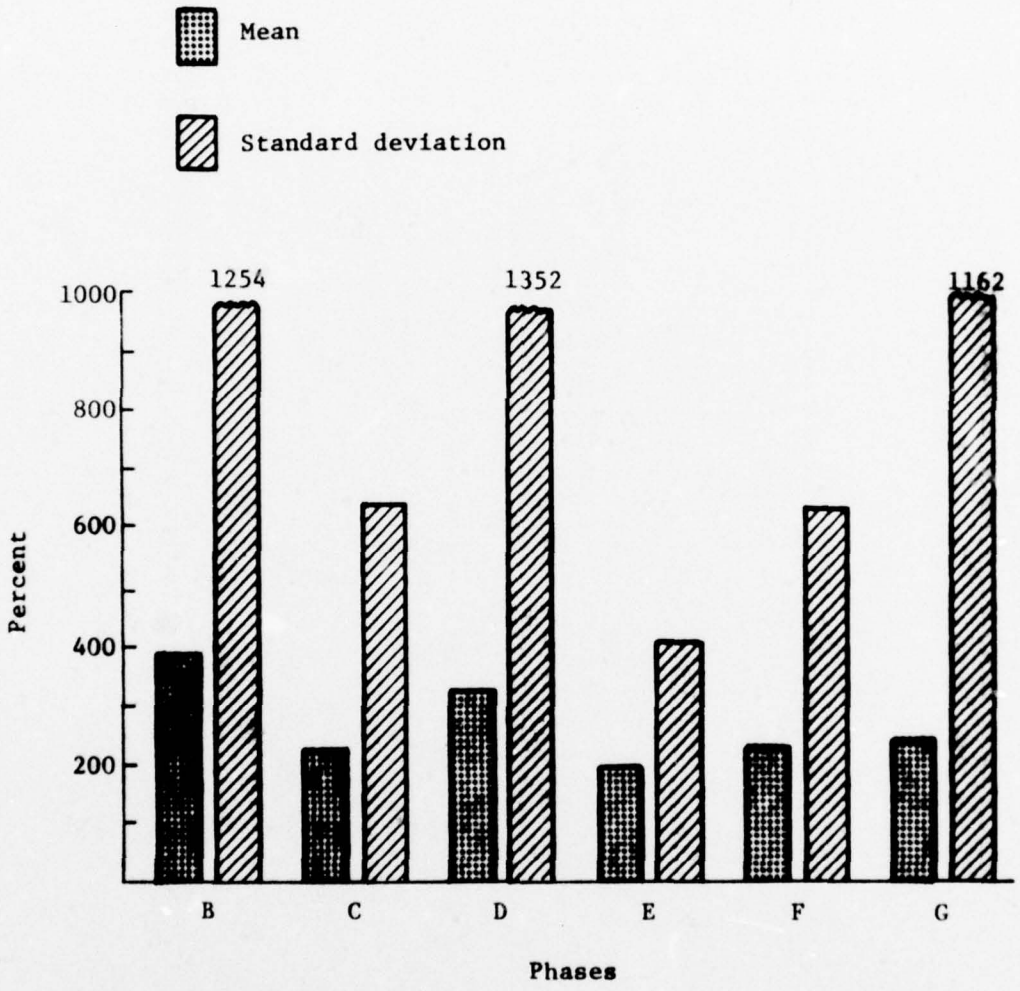


Fig. 9. Mean and standard deviation of percent difference.

Fig. 10. Mean and standard deviation of percentage difference/phase.



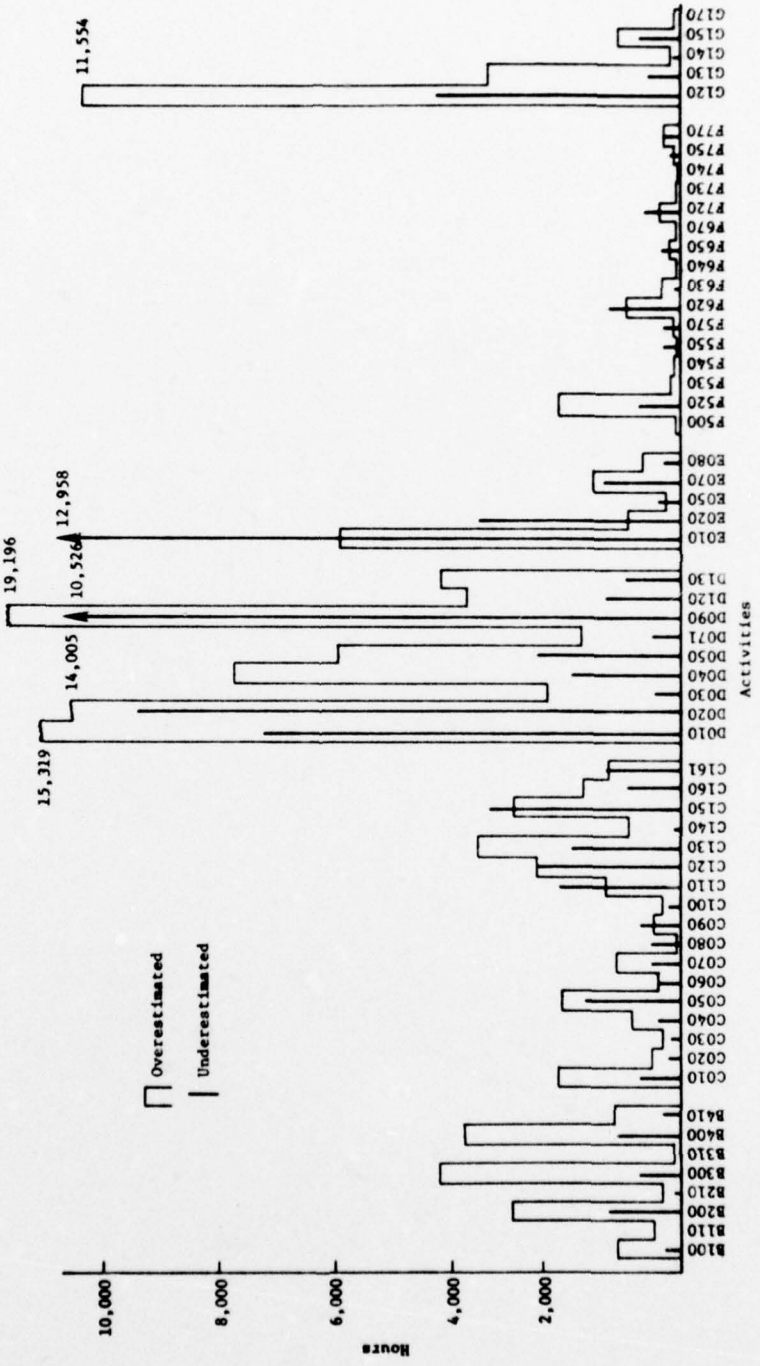


Fig. 11. Hours underestimated versus overestimated.

Fig. 12. Hours underestimated versus overestimated/phase.

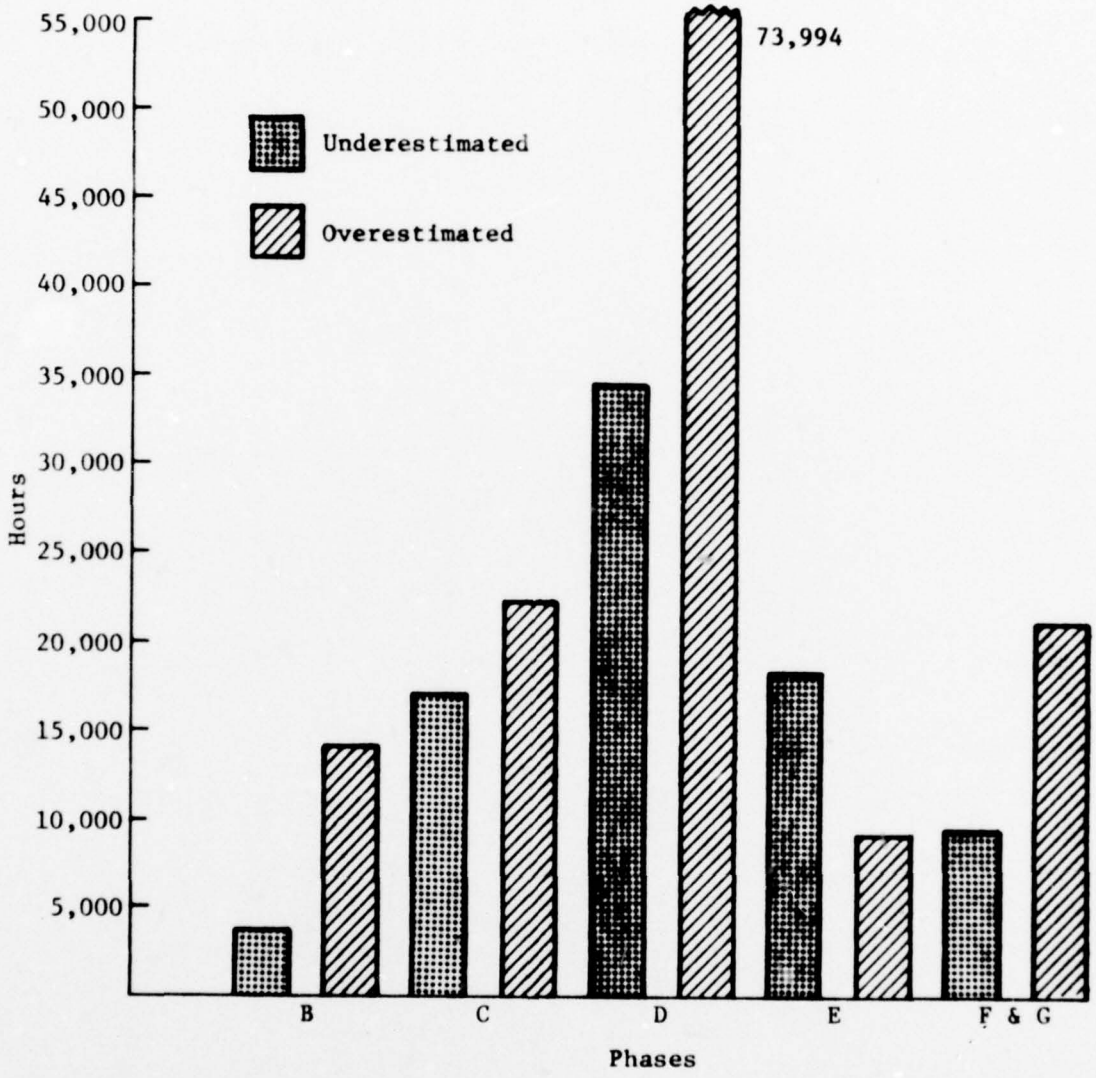
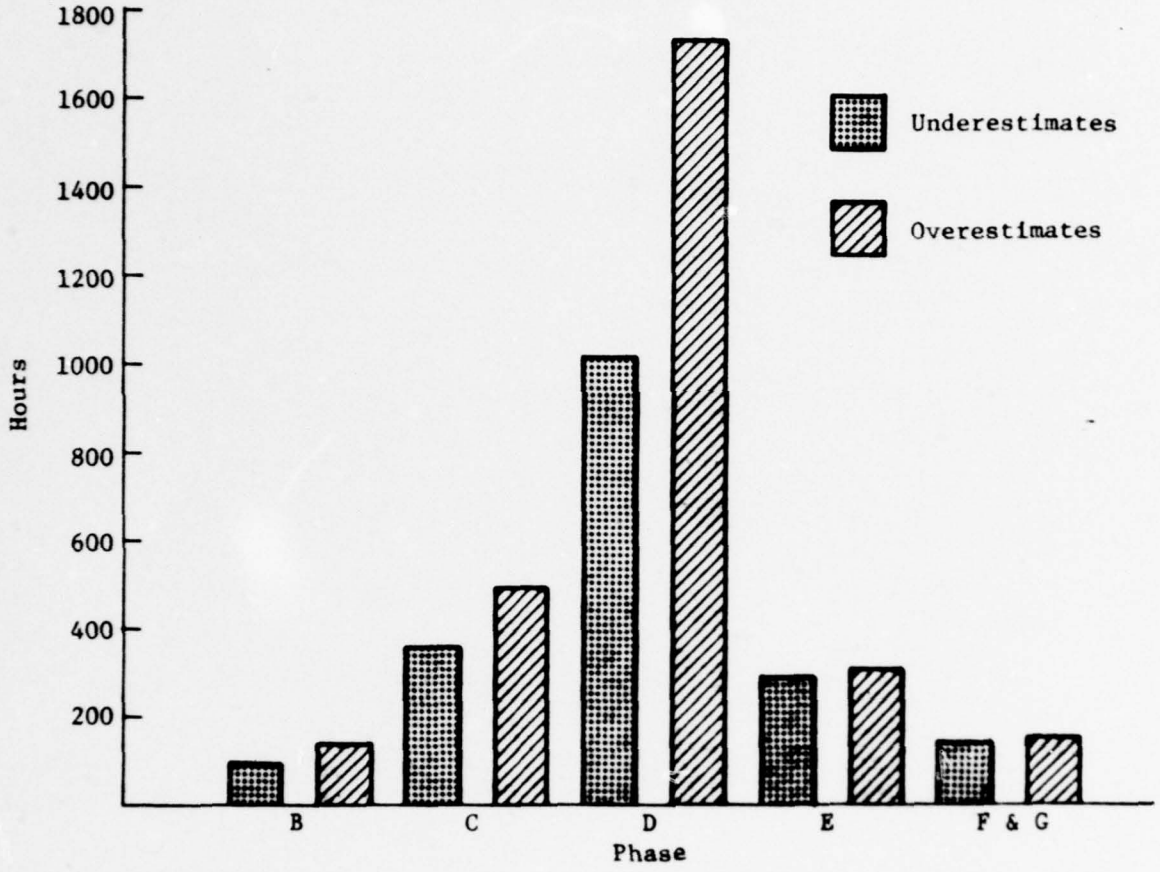


Fig. 13. Number of underestimates versus overestimates/phase.



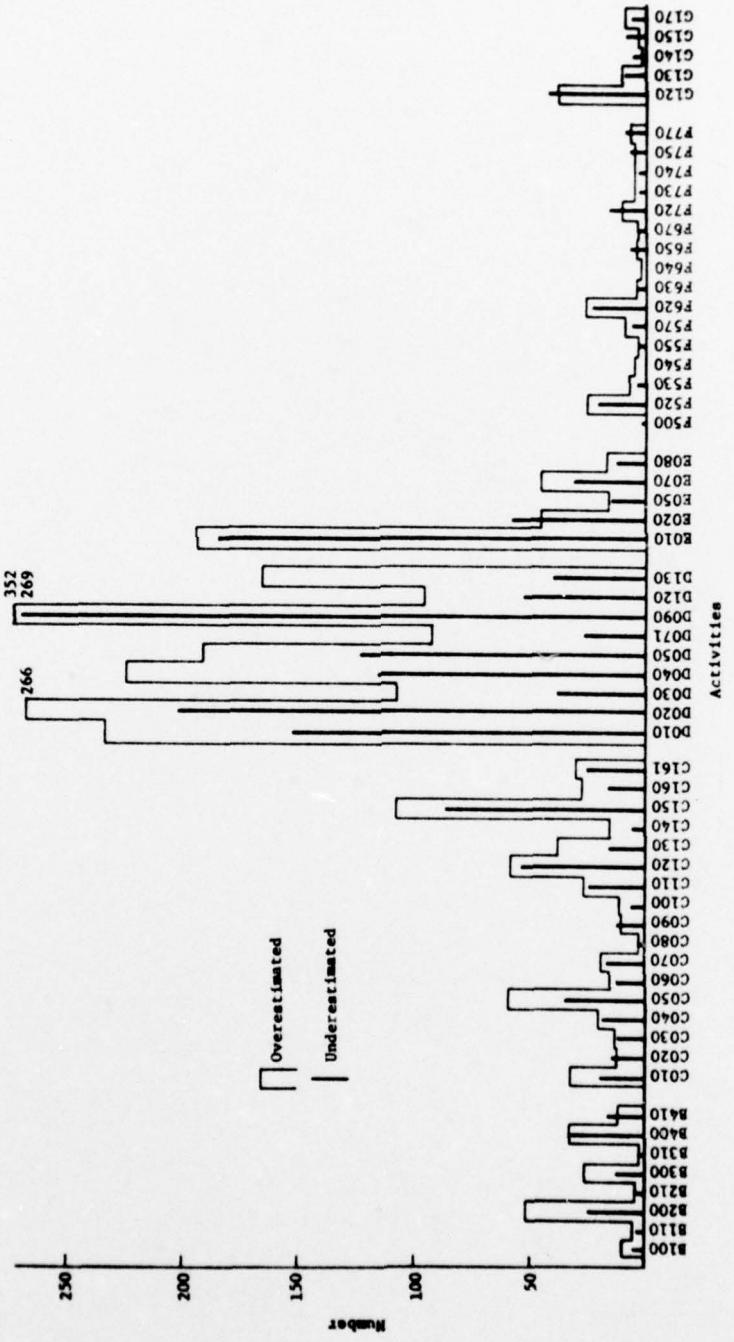


Fig. 14. Number of activities underestimated versus overestimated.

Fig. 15. Functional analyst hours/phase.

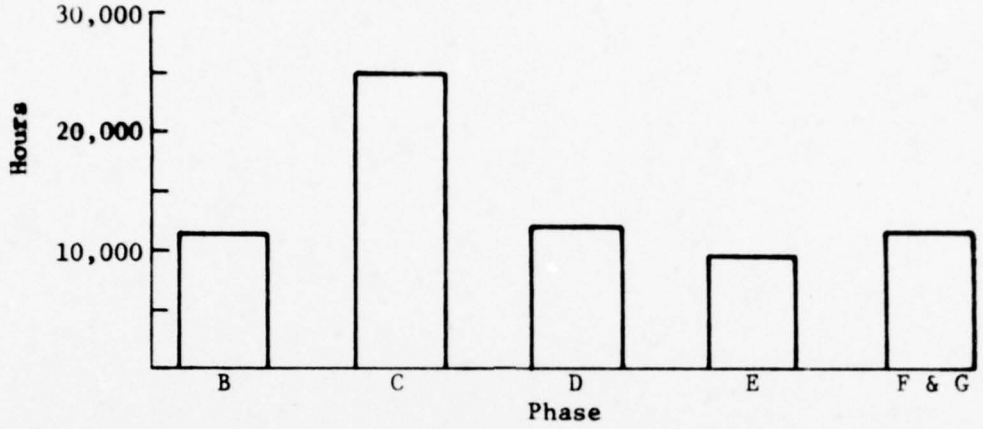


Fig. 16. Data systems analyst hours/phase.

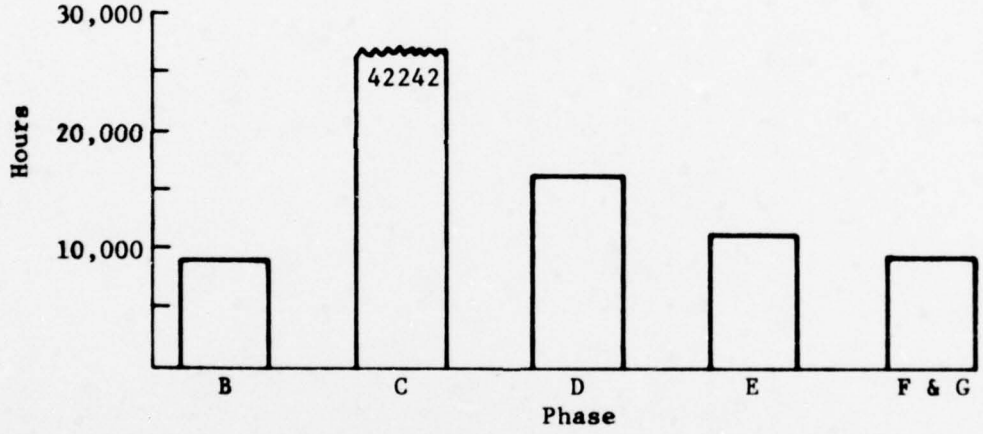


Fig. 17. Programmer hours/phase.

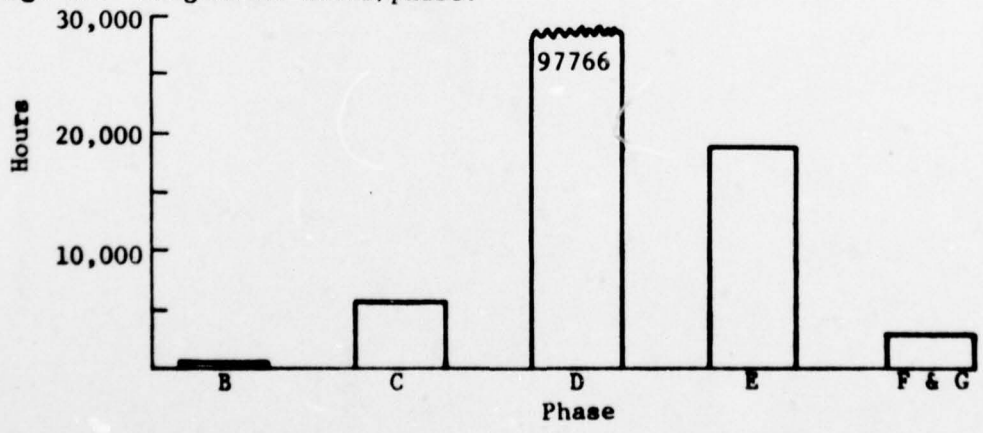


Fig. 18. Hours/skill/phase.

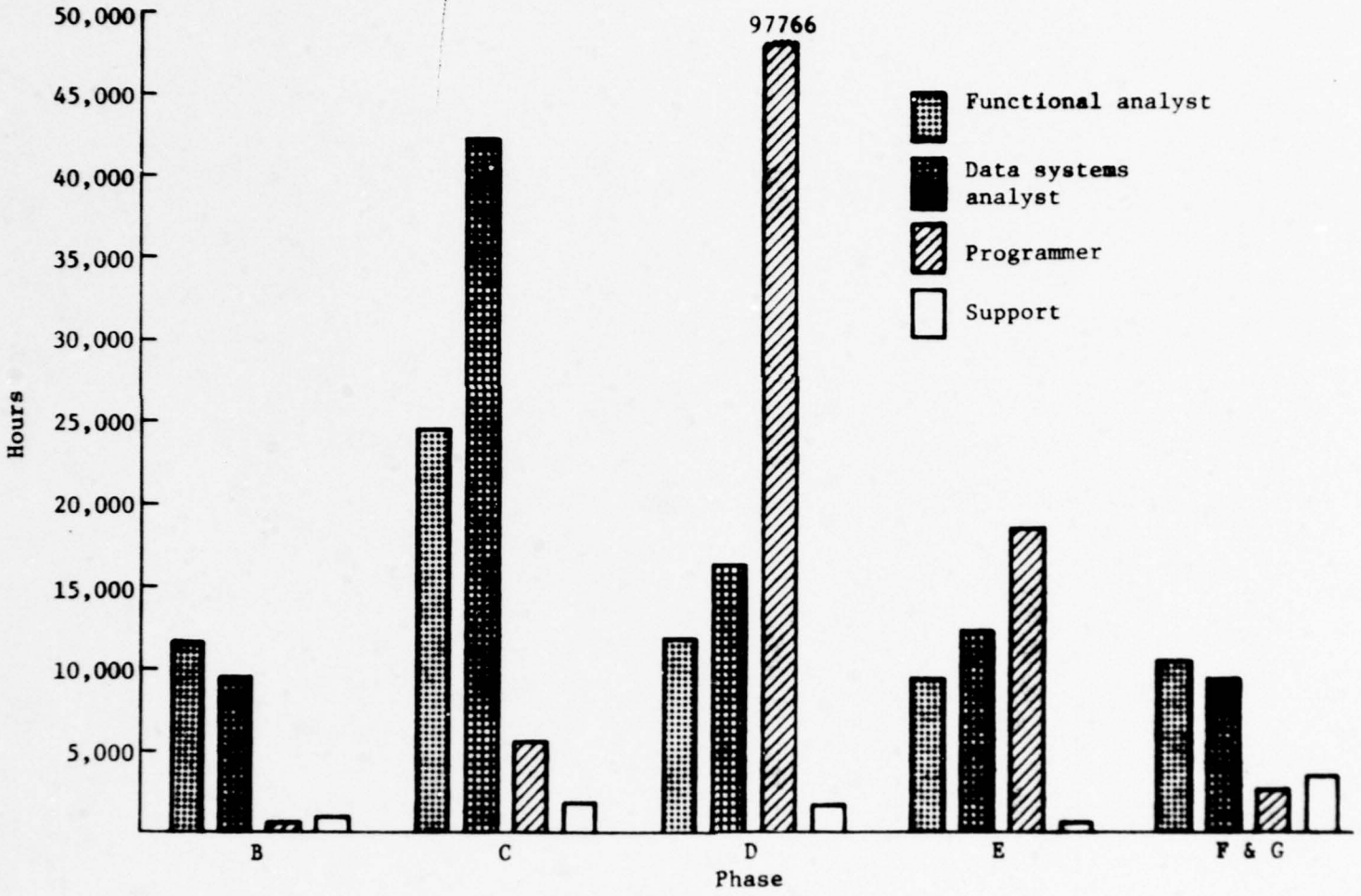


Table 5. Project Summary

Project	Number activities/phase					Hours estimated/expended Percent of total expended					Est/exp totals of selected activities	Est/exp totals of actual project	Rule of thumb Percent of totals expended				Hours expended/percent by skill			
	B	C	D	E	F&G	B	C	D	E	F & G			ANAL	PGM	TEST	DOC	FA	DA	PG	SP
1	3	12	54	9	10	350 / 346 8.8	809 / 713 18.1	1406 / 1632 41.4	484 / 571 14.5	266 / 680 17.3	3315 3942	3736 4243	27	25	23	25	1364 34.6	869 22.0	1591 40.4	118 3.0
2	0	13	33	21	9	0 / 0 0	308 / 357 17.7	624 / 1115 55.3	178 / 224 11.1	162 / 322 16.0	1272 2018	1350 2045	18	41	23	18	193 9.6	619 30.7	1102 54.6	104 5.2
3	0	27	64	18	1	0 / 0 0	741 / 728 28.9	1210 / 1258 49.9	370 / 528 20.9	40 / 4 0.2	2361 2518	3203 3325	29	16	55	0	1616 64.2	96 3.8	784 31.1	22 0.9
4	0	4	12	4	4	0 / 0 0	100 / 124 7.6	955 / 1015 62.2	191 / 175 10.7	317 / 317 19.4	1563 1631	2125 2145	8	42	31	19	198 12.1	322 19.7	1005 61.6	106 6.5
5	0	5	16	19	3	0 / 0 0	360 / 558 21.5	544 / 1232 47.5	705 / 791 30.5	35 / 12 0.46	1644 2593	1974 2752	22	22	56	0	135 5.2	1541 59.4	894 34.5	23 0.9
6	0	52	97	143	16	0 / 0 0	3782 / 5535 35.9	2865 / 2367 15.4	4647 / 6881 44.8	535 / 593 3.9	11829 15376	27096 23420	36	9	51	4	2539 16.5	6821 44.4	5868 38.2	148 1
7	1	17	137	3	18	50 / 48 0.7	1694 / 1772 26.1	3327 / 3145 46.2	620 / 600 8.8	1103 / 1236 18.2	6794 6801	7449 7150	27	45	11	18	10 0.1	2379 34.9	4272 62.8	140 2.1

Table 5. (continued)

Project	Number activities/phase					Hours estimated/expended Percent of total expended					Est/exp totals of selected activities	Est/exp totals of actual project	Rule of thumb Percent of totals expended				Hours expended/percent by skill			
	B	C	D	E	F&G	B	C	D	E	F & G			ANAL	PGM	TEST	DOC	FA	DA	PG	SP
8	0	17	68	5	12	0/0 0	312/271 28.7	365/477 50.5	72/84 8.9	237/112 11.9	986/944	1677/1398	29	42	18	12	0/0	348/36.9	501/53.1	95/10.1
9	0	66	464	103	47	0/0 0	3445/4520 10.5	28421/27612 63.9	4124/4400 10.2	16502/6705 15.5	52492/43237	64631/48477	11	44	29	16	10389/24.0	6814/15.8	25110/58.1	924/2.1
10	0	42	74	32	8	0/0 0	1539/1860 18.5	3140/3370 33.5	1855/4174 41.5	375/656 6.5	6909/10060	10248/13749	19	27	48	7	2854/28.4	2905/28.9	4187/41.6	114/1.1
11	0	59	64	23	7	0/0 0	1751/1541 24.3	2777/1975 31.1	2195/1940 30.6	560/888 13.9	7283/6344	8802/6794	24	20	40	15	1439/22.7	1765/27.8	2985/47.1	155/2.4
12	0	6	7	3	0	0/0 0	110/86 15.5	190/304 54.8	40/165 29.7	0/0 0	340/555	426/659	16	31	53	1	3/0.5	248/44.7	303/54.6	1/0.2
13	0	4	6	4	0	0/0 0	60/58 15.3	200/228 60.2	56/93 24.5	0/0 0	316/379	331/394	15	29	53	3	29/7.7	122/32.2	228/60.2	0/0
14	0	8	15	15	10	0/0 0	127/224 6.4	476/1228 34.9	332/1129 32.1	236/936 26.6	1171/3517	1699/4401	6	22	39	32	357/10.2	663/18.9	2132/60.6	365/10.4

Table 5. (continued)

Project	Number activities/phase					Hours estimated/expended Percent of total expended					Est/exp totals of selected activities	Est/exp totals of actual project	Rule of thumb Percent of totals expended				Hours expended/percent by skill			
	B	C	D	E	F&G	B	C	D	E	F & G			ANAL	PGM	TEST	DOC	FA	DA	PG	SP
15	3	112	405	47	24	400/424 0.9	25725/23868 51.1	19510/15426 33.0	3884/3930 8.4	2888/3064 6.6	52407/46712	64811/64944	52	25	15	7	14314/30.6	14298/30.6	16374/35.1	1726/3.7
16	0	10	25	1	1	0/0 0	125/16 12.3	167/112 86.1	10/1 0.8	10/1 0.8	312/130	423/148	12	63	24	1	9/6.9	0/0	121/93.1	0/0
17	0	3	5	1	1	0/0 0	30/3 21.4	35/9 64.3	5/1 7.1	0/1 7.1	70/14	89/20	21	50	21	7	0/0	2/14.3	12/85.7	0/0
18	0	2	2	4	1	0/0 0	36/58 35.0	61/5 3.0	44/100 60.2	3/3 1.8	144/166	146/166	35	3	60	2	67/40.4	0/0	95/57.2	4/2.4
19	0	28	137	7	6	0/0 0	5515/3835 35.6	4792/3720 34.5	1240/2671 24.8	691/553 5.1	12238/10779	13754/11561	36	16	42	7	24/0.2	5037/46.7	5412/50.2	306/2.8
20	0	32	183	16	9	0/0 0	1734/3079 21.0	7129/7836 53.5	2100/3154 21.5	630/567 3.9	11593/14636	16668/19095	21	25	45	10	0/0	6408/43.8	7759/53.0	469/3.2
21	7	15	58	6	11	316/2438 3.2	1910/1163 55.3	2864/321 26.4	240/344 7.3	150 7.8	5480/4406	6183/4676	59	23	11	8	1959/44.5	1041/23.6	1158/26.3	248/5.6

Table 5. (continued)

Project	Number activities/phase					Hours estimated/expended Percent of total expended					Est/exp totals of selected activities	Est/exp totals of actual project	Rule of thumb Percent of totals expended				Hours expended/percent by skill			
	B	C	D	E	F&G	B	C	D	E	F & G			ANAL	PGM	TEST	DOC	FA	DA	PG	SP
22	0	1	0	3	2	0/0 0	99/13 10.6	0/0 0	130/46 37.4	95/64 52.0	324/123	783/195	11	0	37	52	37/30.1	67/54.5	19/15.4	0/0
23	0	1	0	5	1	0/0 0	99/11 2.7	0/0 0	260/314 77.7	95/79 19.5	454/404	783/645	3	0	78	19	121/29.9	24/5.9	230/56.9	29/7.2
24	0	6	1	1	0	0/0 0	211/49 57.6	10/24 28.2	15/12 14.1	0/0 0	236/85	333/187	58	0	42	0	61/71.8	0/0	24/28.2	0/0
25	1	28	13	15	0	40/48 6.6	192/139 19.3	260/287 39.8	230/247 34.3	0/0 0	722/721	823/721	26	0	74	0	402/55.7	31/4.3	286/39.7	2/0.3
26	0	7	5	4	0	0/0 0	120/48 42.5	106/24 21.2	80/41 36.3	0/0 0	306/113	337/118	43	0	58	0	82/72.6	10/8.8	21/18.6	0/0
27	0	0	36	5	0	0/0 0	0/0 0	419/432 86.7	110/66 13.3	0/0 0	529/498	618/498	0	43	57	0	0/0	140/28.1	348/71.9	0/0
28	0	0	22	4	0	0/0 0	0/0 0	424/358 74.6	200/122 25.4	0/0 0	624/480	804/611	0	57	43	0	0/0	66/13.7	414/86.3	0/0

Table 5. (continued)

Project	Number activities/phase					Hours estimated/expended Percent of total expended					Est/exp totals of selected activities	Est/exp totals of actual project	Rule of thumb Percent of totals expended				Hours expended/percent by skill			
	B	C	D	E	F&G	B	C	D	E	F & G			ANAL	PGM	TEST	DOC	FA	DA	PG	SP
29	0	8	5	10	4	0 0	198 247	71 46	644 548	27 272	940 1113	1251 1286	22	1	49	27	199 17.9	835 75.0	179 16.1	0 0
30	0	1	4	1	5	0 0	320 351	1017 977	420 619	241 260	1998 2207	2203 2522	16	0	72	12	0 0	256 11.6	1779 80.6	172 7.8
31	0	0	13	0	1	0 0	0 0	676 1310	0 0	215 214	891 1524	1477 1724	0	86	0	14	0 0	104 6.8	1420 93.2	0 0
32	0	12	130	0	11	0 0	2105 1843	8605 5389	0 0	2750 456	13460 7688	14400 8817	24	65	5	6	923 12.0	4136 53.8	2589 33.7	40 0.5
33	0	45	109	0	0	0 0	725 773	2923 6041	0 0	0 0	3648 6814	3648 6814	11	28	60	0	1658 24.3	908 13.3	4210 61.8	38 0.5
34	0	0	8	4	3	0 0	0 0	80 79	30 13	40 47	150 139	179 162	0	42	24	34	32 23.0	28 20.1	79 56.8	0 0
35	0	8	8	0	2	0 0	215 249	350 537	0 0	200 103	765 889	1309 1381	28	35	26	12	8 0.9	289 32.5	567 63.8	25 2.8

Table 5. (continued)

Project	Number activities/phase					Hours estimated/expended Percent of total expended					Est/exp totals of selected activities	Est/exp totals of actual project	Rule of thumb Percent of totals expended				Hours expended/percent by skill			
	B	C	D	E	F&G	B	C	D	E	F & G			ANAL	PGM	TEST	DOC	FA	DA	PG	SP
36	6	3	0	7	4	500 / 530 27.7	225 / 445 23.3	0 / 0 0	310 / 322 16.9	1030 / 613 32.1	2065 1910	2123 1948	51	0	17	32	139 / 7.3	1179 / 61.7	326 / 17.1	266 / 13.9
37	0	4	5	0	0	0 / 0 0	160 / 76 31.9	203 / 162 68.1	0 / 0 0	0 / 0 0	363 238	363 238	32	41	27	0	17 / 7.1	74 / 31.1	138 / 58.0	9 / 3.8
38	0	4	27	0	0	0 / 0 0	560 / 135 24.5	280 / 416 75.5	0 / 0 0	0 / 0 0	840 551	914 551	24	56	20	0	4 / 0.7	182 / 33.0	361 / 65.5	4 / 0.7
39	0	20	166	26	12	0 / 0 0	8649 / 7006 31.5	13698 / 10187 45.9	3838 / 3212 14.5	2752 / 1803 8.1	28937 22208	32686 23289	32	30	30	8	77 / 0.3	9126 / 41.1	12344 / 55.6	661 / 3.0

the distribution of resources in software development, the columns titled "Rules of Thumb Percent of Totals Expended" were added. These figures were arrived at by slightly reclassifying activities in the following way:

Analysis	- sum of phases B and C	
Programming	- sum of activities	D010 D020 D030 D040 D050
Testing	- sum of activities	D071 D090 D120 and Phase E
Documentation	- sum of activity	D130 and Phases F and G

Figures 19 through 26 are the comparative (graphical) representations of the data in Table 5.

The number of unique activities - phases scored per project, in relation to the criteria specified, are depicted in Tables 6 and 7.

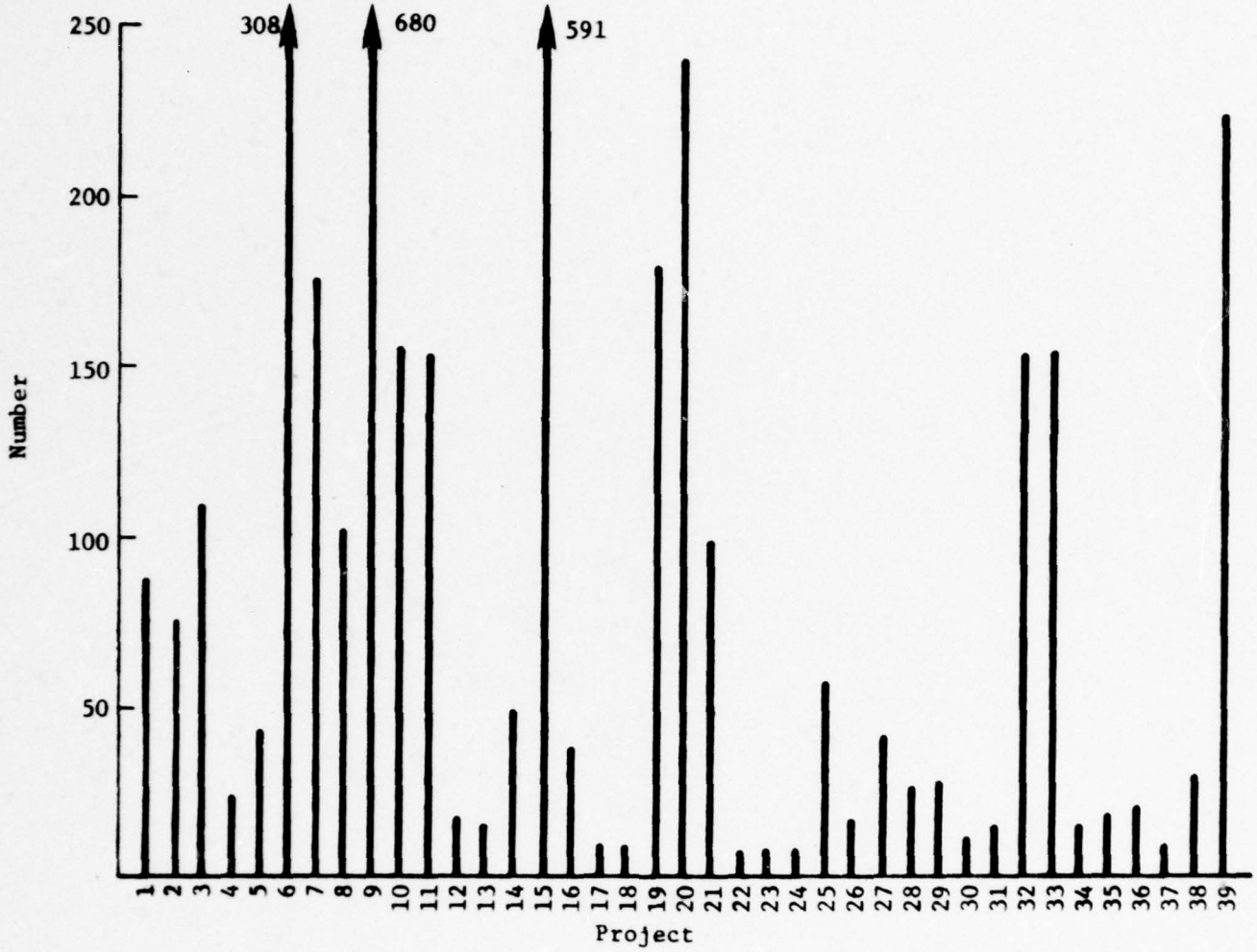
Tables 8 and 9 contain the most pertinent data extracted from the 24 outputs from SEQUIN. (A complete sample output from one of the SEQUIN runs used by this research is contained in Appendix 3.) These two tables represent changes in the scoring criterion which specified:

(a) a unique activity within a project is scored 1 if at least half (1/2) of those activities (with identical activity codes) fall within the range of some variable (x) times the standard deviation (SD). Refer to algorithm on page 81.

(b) a unique activity within a project is scored 1 if at least three fourths (3/4) of those activities (with identical activity codes) fall within the range of some variable (x) times the standard deviation (SD). Refer to algorithm on page 81.

The columns on these tables correspond to the SEQUIN output resulting from changes to the variable multiplier used to adjust the range of

Fig. 19. Number of activities/project.



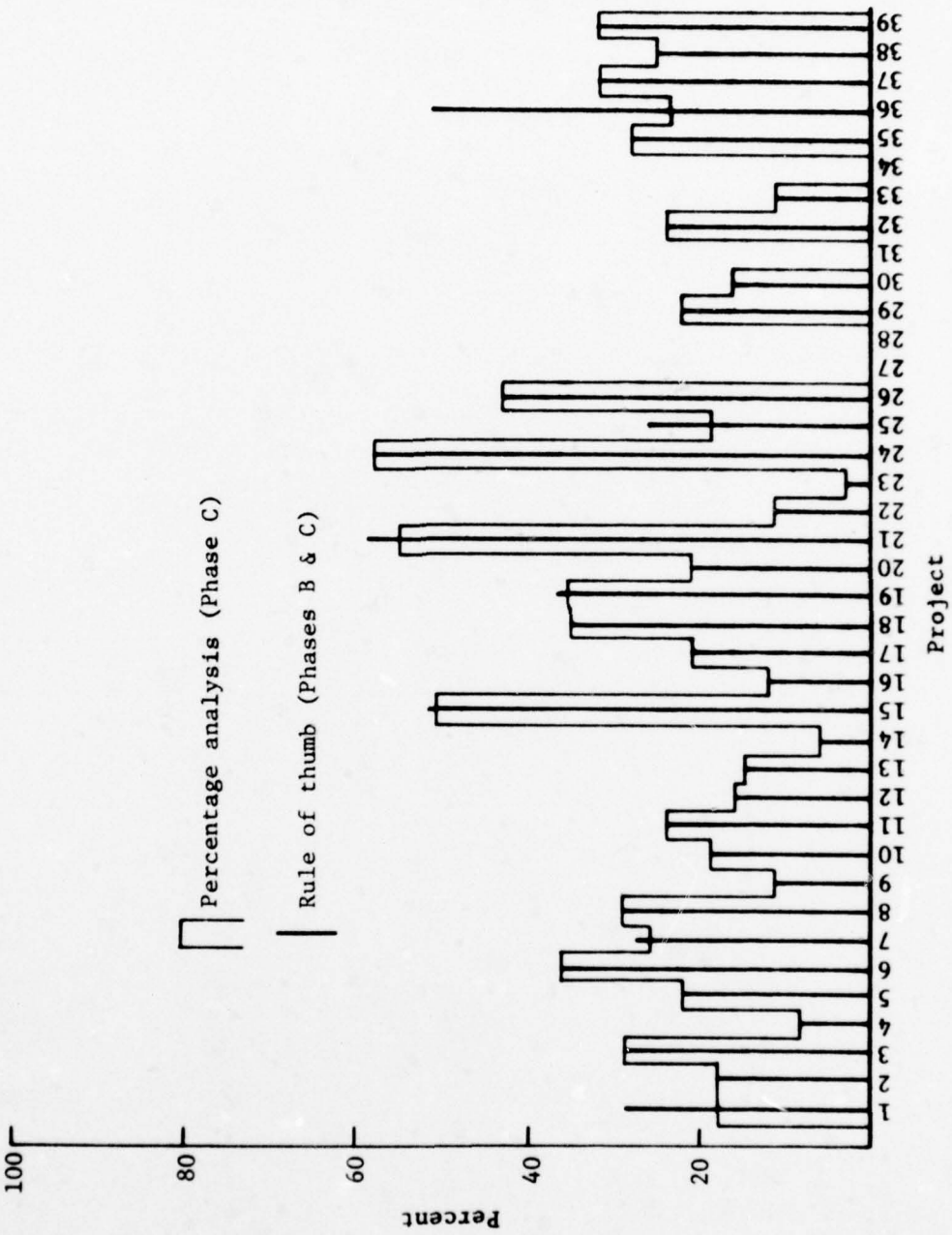


Fig. 20. Percentage analysis (Phase C)/project versus rule of thumb percentage analysis (Phases B & C)/project.

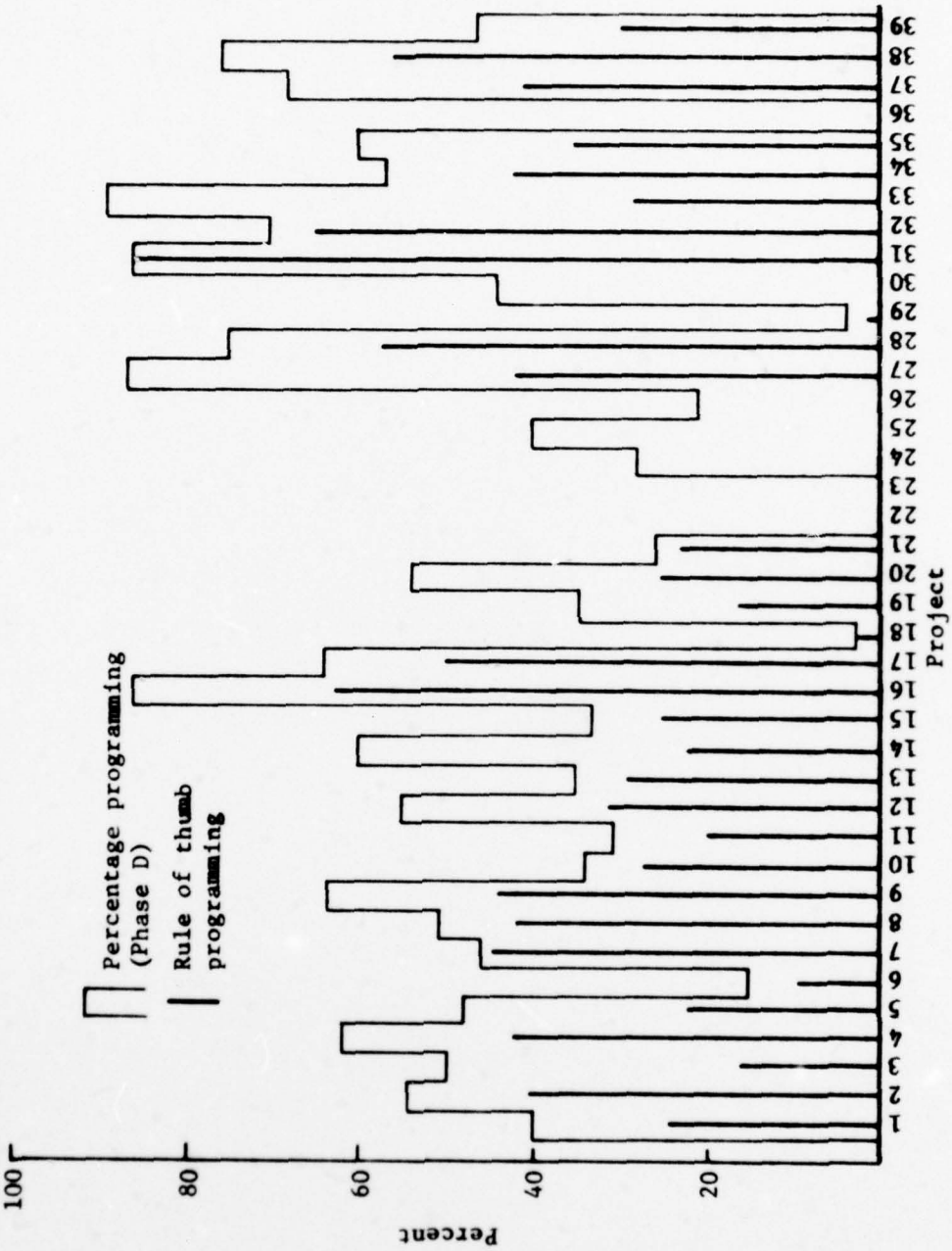


Fig. 21. Percentage programming (Phase D)/project versus rule of thumb percentage programming/project.

Fig. 22. Percentage testing (Phase E)/project versus rule of thumb percentage testing/project.

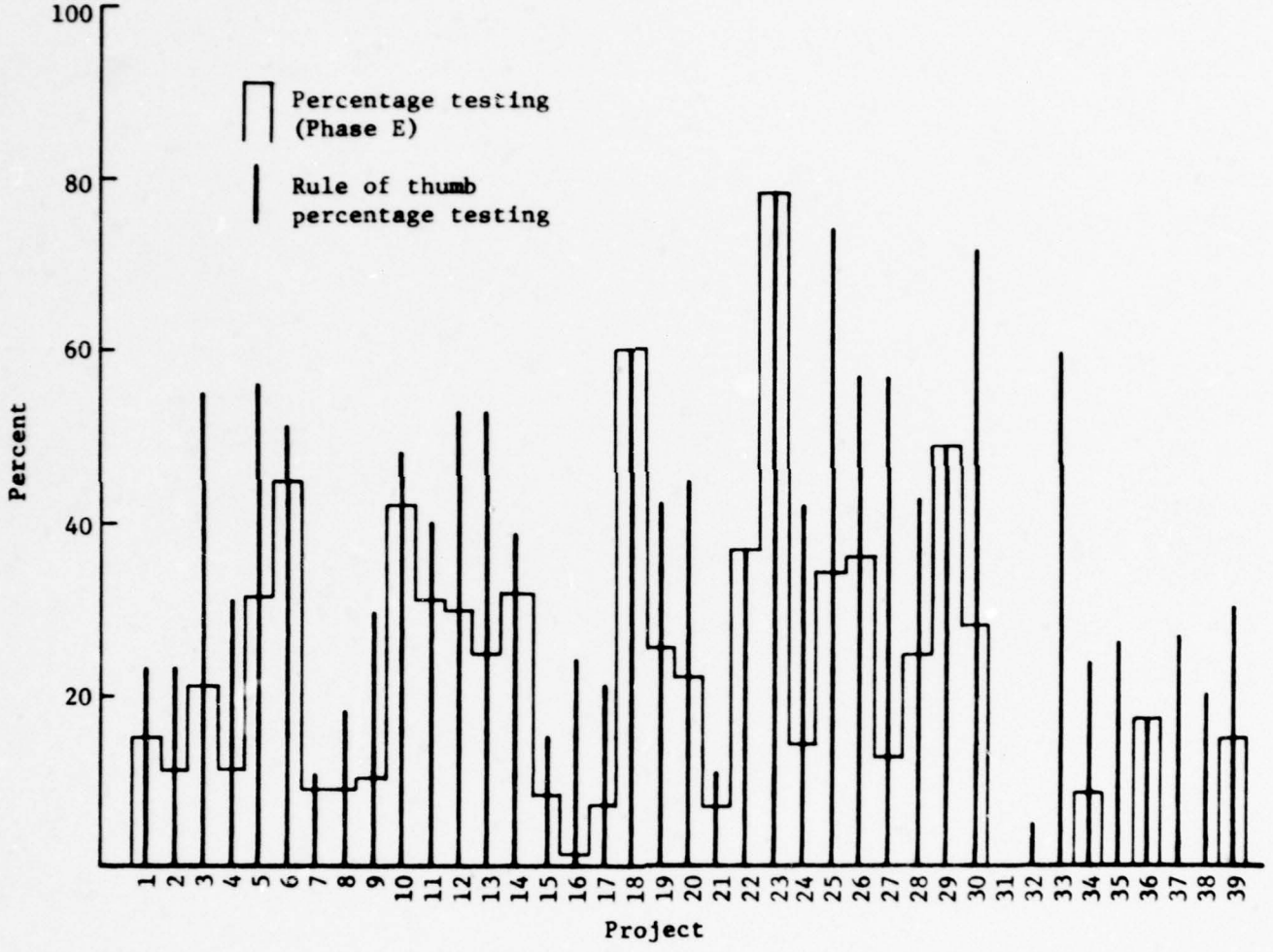
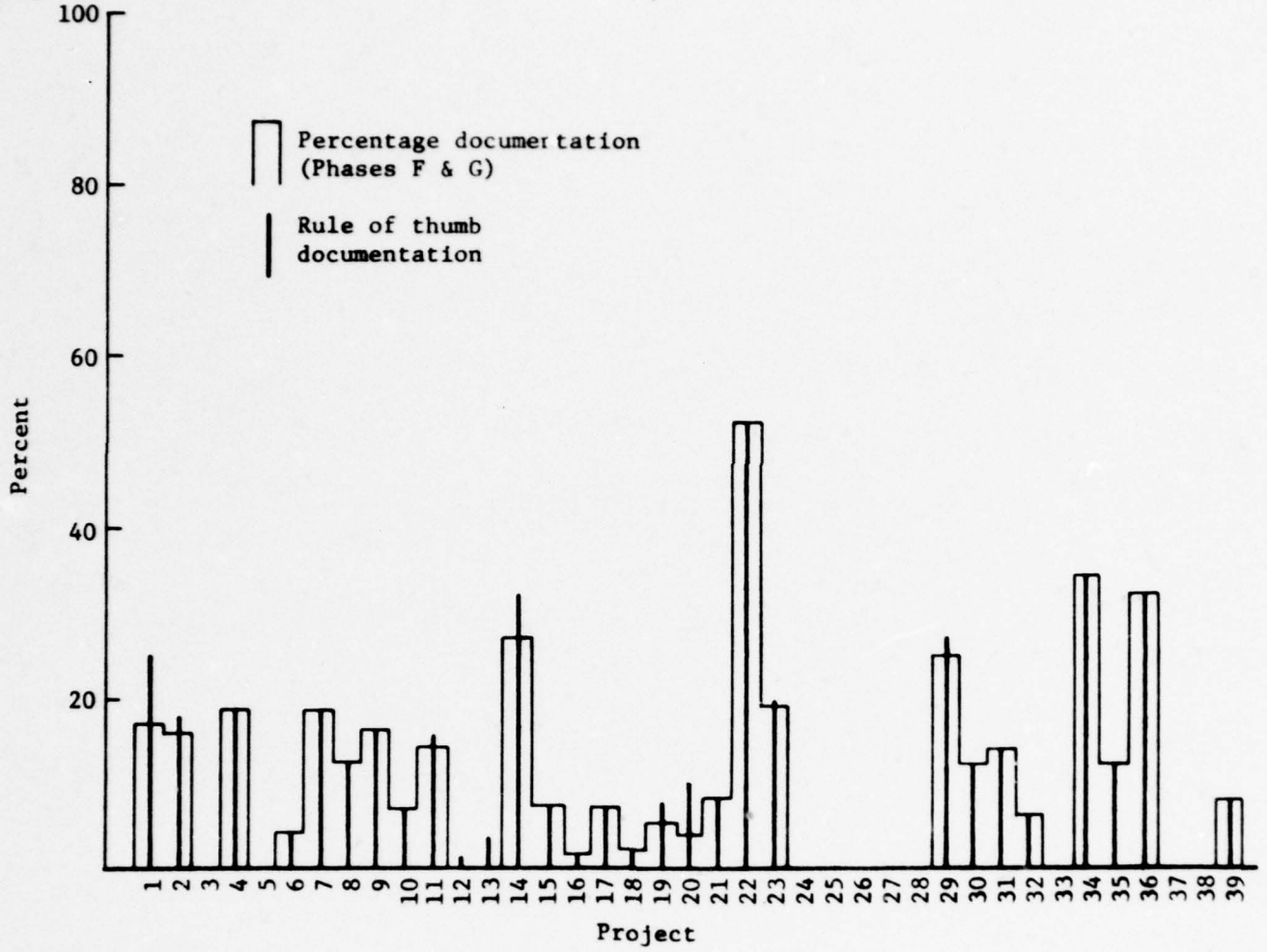


Fig. 23. Percentage documentation (Phases F & G)/project versus rule of thumb documentation/project.



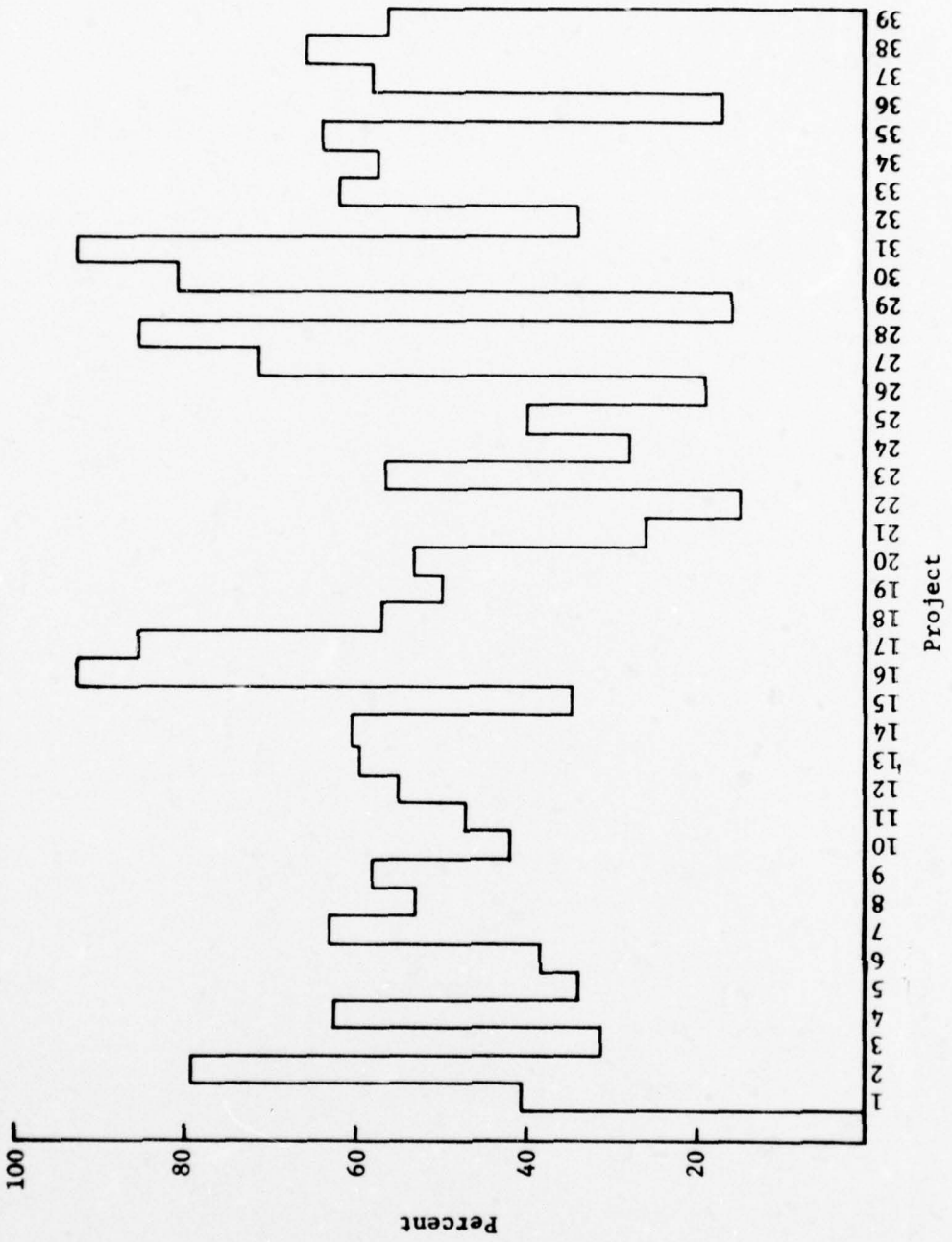


Fig. 24. Percentage programmer skill/project.

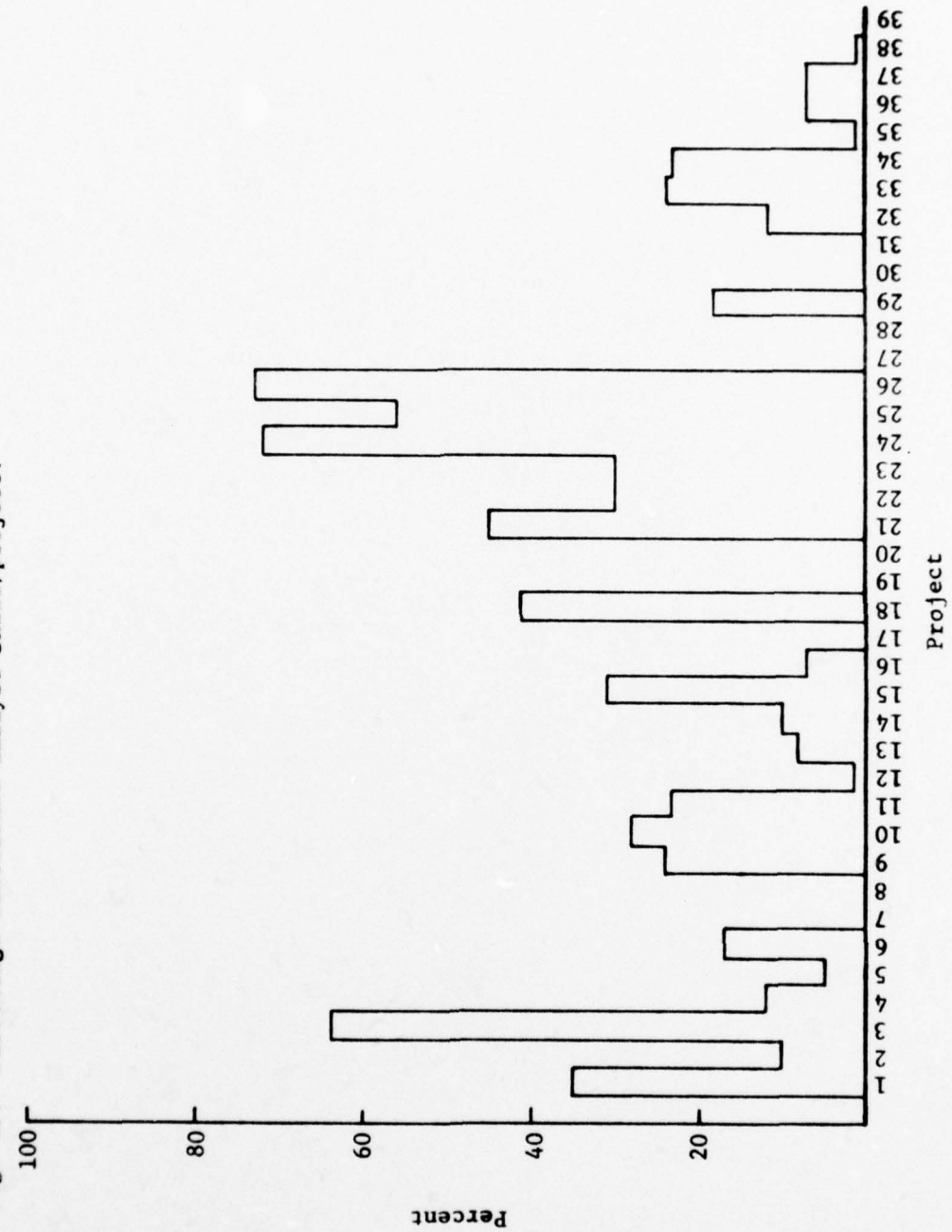


Fig. 25. Percentage functional analyst skill/project.

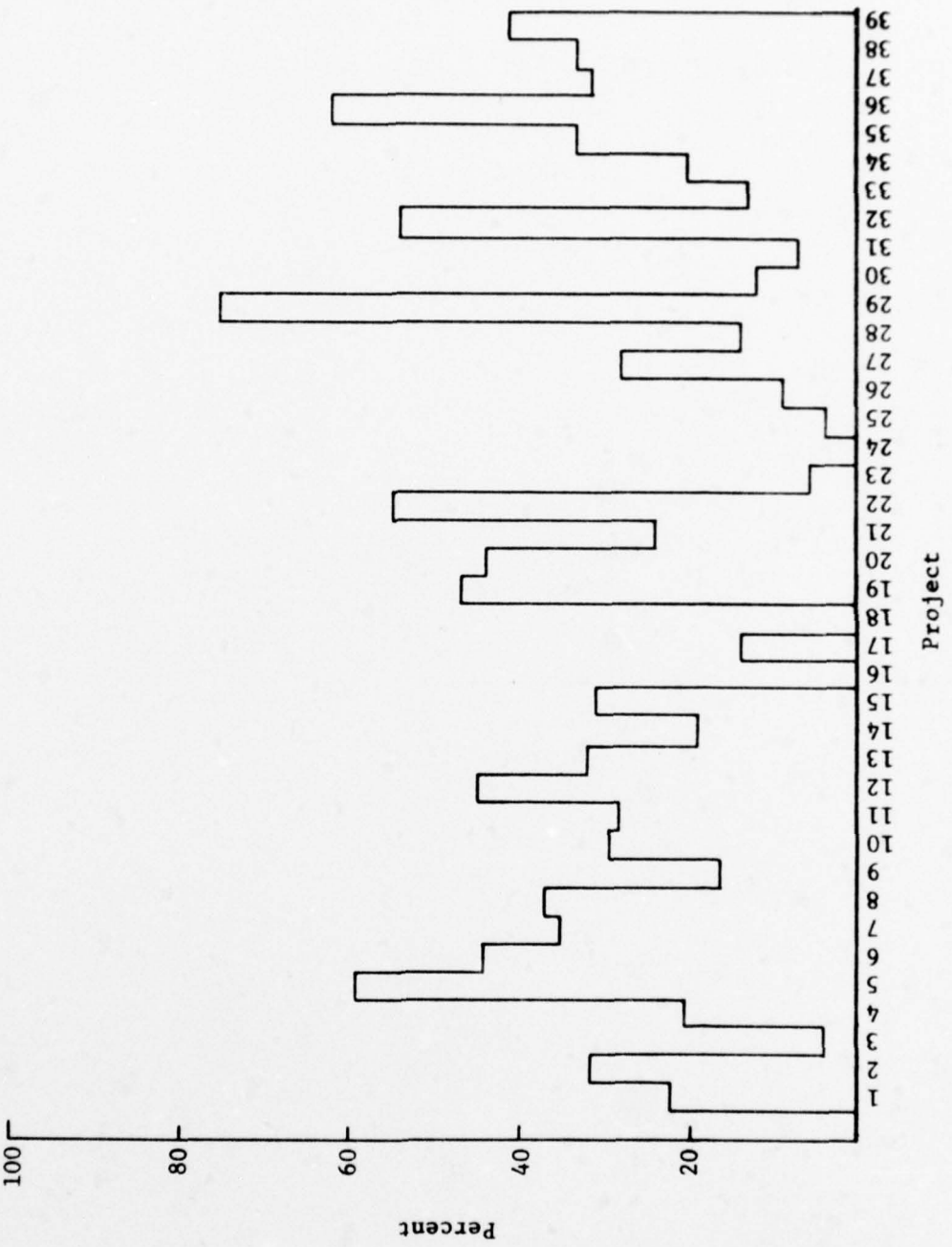


Fig. 26. Percentage data analyst skill/project.

Table 6. Number of Unique Activities - Phases/Project Arithmetic Difference

Project	Number of unique activities-phases	1/2 within x · SD						3/4 within x · SD					
		0.1	0.25	1.0	1.25	1.50	2.0	0.1	0.25	1.0	1.25	1.50	2.0
1	29-6	19-0	24-1	29-3	29-4	29-4	29-5	15-0	21-1	28-3	28-4	28-4	29-5
2	23-5	10-1	15-1	21-1	22-1	23-1	23-1	6-1	12-1	20-1	21-1	22-1	23-1
3	9-4	4-1	8-1	9-1	9-1	9-1	9-1	2-1	7-1	8-1	8-1	8-1	9-1
4	6-5	5-1	6-1	6-4	6-4	6-4	6-4	2-1	6-1	6-4	6-4	6-4	6-4
5	7-4	0-0	3-0	7-1	7-1	7-1	7-1	0-0	2-0	7-1	7-1	7-1	7-1
6	20-4	4-0	12-0	18-0	19-0	20-0	20-0	2-0	5-0	16-0	18-0	19-0	19-0
7	35-6	18-2	28-2	31-3	33-3	33-3	34-3	14-2	24-2	31-3	33-3	33-3	33-3
8	35-5	18-1	24-1	30-2	32-2	33-2	34-3	16-1	23-1	30-2	33-2	33-2	34-3
9	39-5	7-0	15-0	31-0	32-0	33-0	33-0	2-0	6-0	23-0	25-0	28-0	29-0
10	27-5	11-0	19-0	25-1	26-1	26-2	27-2	8-0	16-0	22-1	24-1	24-2	26-2
11	23-5	6-0	14-0	22-1	22-1	22-1	22-1	4-0	9-0	21-1	22-1	22-1	22-1
12	13-3	8-0	10-0	12-1	13-1	13-1	13-3	7-0	8-0	12-1	13-1	13-1	13-3
13	12-3	4-0	9-0	12-1	12-2	12-3	12-3	3-0	8-0	12-1	12-2	12-3	12-3
14	13-5	5-0	7-0	11-1	12-1	12-2	13-2	3-0	4-0	7-1	11-1	12-2	13-2
15	43-6	10-0	21-0	38-1	39-1	41-1	42-2	2-0	11-0	30-1	34-1	37-1	40-2
16	6-4	3-0	6-2	6-3	6-3	6-4	6-4	3-0	6-2	6-3	6-3	6-4	6-4
17	6-4	6-2	6-2	6-4	6-4	6-4	6-4	5-2	6-2	6-4	6-4	6-4	6-4
18	4-4	4-1	4-1	4-3	4-4	4-4	4-4	2-1	3-1	4-3	4-4	4-4	4-4
19	18-4	3-0	7-0	13-0	13-0	14-0	15-0	2-0	4-0	10-0	11-0	12-0	13-0
20	24-5	2-0	7-0	17-1	17-1	19-1	22-1	1-0	3-0	13-1	14-1	16-1	20-1

Table 6. (continued)

Project	Number of unique activities-phases	1/2 within x · SD						3/4 within x · SD					
		0.1	0.25	1.0	1.25	1.50	2.0	0.1	0.25	1.0	1.25	1.50	2.0
21	34-6	12-0	20-1	30-1	33-2	33-2	33-4	8-0	15-1	26-1	28-2	31-2	32-4
22	5-3	0-0	0-0	3-0	4-2	4-3	5-3	0-0	0-0	3-0	4-2	4-3	5-3
23	4-3	0-0	0-0	3-1	4-2	4-2	4-2	0-0	0-0	3-1	4-2	4-2	4-2
24	8-3	2-1	4-2	8-2	8-2	8-2	8-3	2-1	4-2	8-2	8-2	8-2	8-3
25	5-4	5-1	5-1	5-1	5-2	5-2	5-3	3-1	5-1	5-1	5-2	5-2	5-3
26	5-3	1-0	5-0	5-3	5-3	5-3	5-3	1-0	5-0	5-3	5-3	5-3	5-3
27	8-2	5-0	8-0	8-1	8-1	8-1	8-1	2-0	7-0	8-1	8-1	8-1	8-1
28	4-2	2-0	3-0	4-0	4-1	4-1	4-2	2-0	2-0	4-0	4-1	4-1	4-2
29	10-4	3-0	7-0	7-1	7-2	7-2	7-2	1-0	4-0	7-1	7-2	7-2	7-2
30	8-4	1-0	3-0	7-1	7-1	7-1	7-1	1-0	3-0	7-1	7-1	7-1	7-1
31	4-2	2-1	3-1	4-1	4-1	4-1	4-1	2-1	2-1	4-1	4-1	4-1	4-1
32	11-3	0-0	5-0	11-0	11-0	11-0	11-0	0-0	4-0	10-0	11-0	11-0	11-0
33	5-2	3-0	5-0	5-0	5-0	5-0	5-0	2-0	3-0	5-0	5-0	5-0	5-0
34	11-4	8-1	11-2	11-4	11-4	11-4	11-4	7-1	11-2	11-4	11-4	11-4	11-4
35	8-3	2-0	3-0	8-0	8-0	8-0	8-2	0-0	1-0	5-0	7-0	8-0	8-2
36	10-4	4-0	5-0	8-1	9-1	9-1	10-2	4-0	5-0	8-1	8-1	9-1	10-2
37	9-2	1-0	4-0	9-1	9-1	9-2	9-2	1-0	4-0	9-1	9-1	9-2	9-2
38	9-2	5-0	6-0	8-0	9-0	9-0	9-0	3-0	5-0	8-0	8-0	8-0	8-0
39	11-4	0-0	0-0	7-0	7-0	8-0	9-0	0-0	0-0	4-0	6-0	6-0	8-0
Average activities & phases	18.282	5.538	9.256	14.077	14.769	15.205	15.846	3.872	7.256	12.872	13.974	14.564	15.436

Table 7. Number of Unique Activities - Phases/Project Percent Difference

Project	Number of unique activities-phases	1/2 within x · SD						3/4 within x · SD					
		0.1	0.25	1.0	1.25	1.50	2.0	0.1	0.25	1.0	1.25	1.50	2.0
1	29-6	28-1	29-3	29-5	29-5	29-5	29-5	23-1	26-3	29-5	29-5	29-5	29-5
2	23-5	14-0	17-1	23-1	23-1	23-1	23-1	11-0	12-1	19-1	19-1	20-1	20-1
3	9-4	5-0	8-0	9-1	9-1	9-1	9-1	3-0	5-0	9-1	9-1	9-1	9-1
4	6-5	6-2	6-3	6-5	6-5	6-5	6-5	6-2	6-3	6-5	6-5	6-5	6-5
5	7-4	5-0	6-0	6-2	6-3	7-3	7-3	3-0	5-0	6-2	6-3	6-3	6-3
6	20-4	11-0	16-0	20-0	20-0	20-0	20-0	4-0	10-0	19-0	19-0	19-0	19-0
7	35-6	27-2	30-3	32-3	33-3	33-3	34-4	21-2	27-3	32-3	33-3	33-3	34-4
8	35-5	16-0	23-0	30-1	30-1	30-1	32-3	12-0	22-0	28-1	28-1	29-1	32-3
9	39-5	17-0	33-0	35-0	35-0	35-0	35-0	7-0	22-0	33-0	35-0	35-0	35-0
10	27-5	18-0	22-1	26-2	26-2	27-2	27-2	14-0	19-1	25-2	25-2	25-2	25-2
11	23-5	11-1	20-1	23-2	23-2	23-2	23-2	5-1	13-1	23-2	23-2	23-2	23-2
12	13-3	10-0	11-0	13-3	13-3	13-3	13-3	10-0	11-0	13-3	13-3	13-3	13-3
13	12-3	4-0	8-1	12-2	12-2	12-2	12-2	4-0	8-1	12-2	12-2	12-2	12-2
14	13-5	10-0	12-1	13-4	13-4	13-4	13-4	7-0	11-1	13-4	13-4	13-4	13-4
15	43-6	30-0	33-1	41-1	42-1	42-1	42-1	19-0	28-1	39-1	40-1	40-1	40-1
16	6-4	2-0	3-0	4-0	4-0	4-1	4-1	2-0	3-0	4-0	4-0	4-1	4-1
17	6-4	1-0	3-1	4-2	5-2	6-2	6-3	0-0	1-1	4-2	5-2	5-2	5-3
18	4-4	3-1	3-2	4-2	4-2	4-4	4-4	1-1	2-2	4-2	4-2	4-4	4-4
19	18-4	10-0	14-0	18-2	18-2	18-2	18-2	6-0	11-0	17-2	17-2	17-2	17-2
20	24-5	9-0	14-0	23-0	23-1	23-1	23-2	3-0	11-0	22-0	22-1	22-1	23-2

Table 7. (continued)

Project	Number of unique activities-phases	1/2 within x · SD						3/4 within x · SD					
		0.1	0.25	1.0	1.25	1.50	2.0	0.1	0.25	1.0	1.25	1.50	2.0
21	34-6	25-0	28-1	33-5	33-5	34-5	34-5	21-0	23-1	30-5	31-5	31-5	32-5
22	5-3	0-0	1-0	2-1	3-2	3-2	3-3	0-0	0-0	2-1	3-2	3-2	3-3
23	4-3	1-1	3-1	3-1	3-1	4-2	4-3	0-1	2-1	2-1	3-1	4-2	4-3
24	8-3	2-2	2-2	6-2	6-2	6-2	7-2	2-2	2-2	6-2	6-2	6-2	7-2
25	5-4	2-1	4-1	5-1	5-1	5-2	5-2	2-1	3-1	4-1	5-1	5-2	5-2
26	5-3	0-0	4-0	5-0	5-0	5-0	5-1	0-0	2-0	3-0	4-0	5-0	5-1
27	8-2	5-0	6-0	8-0	8-0	8-0	8-0	5-0	5-0	8-0	8-0	8-0	8-0
28	4-2	3-0	3-0	4-0	4-1	4-1	4-1	1-0	2-0	4-0	4-1	4-1	4-1
29	10-4	4-0	6-0	8-1	9-3	9-3	10-3	1-0	5-0	8-1	9-3	9-3	10-3
30	8-4	7-2	7-3	8-4	8-4	8-4	8-4	7-2	7-3	8-4	8-4	8-4	8-4
31	4-2	4-1	4-1	4-1	4-1	4-1	4-2	3-1	4-1	4-1	4-1	4-1	4-2
32	11-3	5-0	7-0	9-0	10-0	10-0	10-1	3-0	4-0	7-0	8-0	8-0	10-1
33	5-2	4-0	5-0	5-0	5-0	5-0	5-0	3-0	5-0	5-0	5-0	5-0	5-0
34	11-4	6-1	7-2	11-3	11-3	11-3	11-4	6-1	7-2	11-3	11-3	11-3	11-4
35	8-3	6-0	7-0	8-3	8-3	8-3	8-3	2-0	5-0	8-3	8-3	8-3	8-3
36	10-4	9-1	9-2	10-2	10-3	10-4	10-4	7-2	8-2	10-2	10-3	10-4	10-4
37	9-2	4-0	7-0	9-2	9-2	9-2	9-2	4-0	7-0	9-2	9-2	9-2	9-2
38	9-2	7-0	8-0	9-0	9-0	9-0	9-1	6-0	8-0	8-0	8-0	8-0	8-1
39	11-4	3-0	9-0	11-0	11-0	11-0	11-0	1-0	3-0	10-0	11-0	11-0	11-0
Average activities & phases	18.282	8.974	12.026	15.205	15.538	15.821	16.256	6.462	9.897	14.564	15.077	15.333	15.897

Table 8. Summarized SEQUIN Results
1/2 of Unique Activities/Project Within x · SD (Scored 1)

	Arithmetic						Percentage					
x · SD	0.1	0.25	1.0	1.25	1.50	2.0	0.1	0.25	1.0	1.25	1.50	2.0
Intercriterion correlation	0.022	-0.087	-0.186	-0.193	-0.203	-0.216	-0.200	-0.244	-0.228	-0.226	-0.219	-0.216
Evaluation mean	5.564	9.256	14.077	14.769	15.205	15.846	8.974	12.026	15.205	15.538	15.821	16.256
Selected activities	16 .799 .179	16 .754 .205	26 .894 .205	26 .890 .231	26 .882 .231	16 .833 .282	16 .791 .231	26 .803 .231	16 .847 .282	16 .842 .282	16 .843 .282	16 .841 .282
	63 .050 .051	45 .114 .179	62 .024 .179	62 .046 .179	62 .055 .179	19 .071 .128	45 .100 .256	16 .081 .256	15 .071 .205	15 .062 .231	15 .064 .231	15 .070 .231
	41 .035 .051	18 .035 .051	48 .036 .077	17 .026 .026	17 .022 .026	26 .038 .256	23 .037 .103	18 .044 .077	46 .024 .128	46 .031 .128	46 .032 .128	62 .026 .179
	52 .028 .051	17 .012 .026	16 .009 .256	16 .006 .256	16 .003 .256	51 .020 .128	30 .023 .231	64 .017 .051	30 .015 .410	30 .019 .410	30 .016 .410	30 .016 .410
	64 .003 .026	15 .006 .179	17 -.002 .026	48 -.001 .077	53 .004 .051	61 -.000 .410	26 .018 .205	61 .010 .333	26 .010 .231	26 .012 .231	26 .012 .256	17 .012 .051
	12 -.003 .026	34 .012 .692	61 -.000 .410	61 .003 .410	32 .000 .462	53 .006 .051	40 .010 .128	13 .007 .282	39 .009 .282	39 .006 .282	39 .008 .282	26 .007 .256
	40 .001 .103	7 .012 .077	15 .006 .205	15 .003 .205	13 .012 .308	52 .005 .077	39 -.000 .077	45 .008 .282	44 .004 .026	61 .004 .410	61 .003 .410	46 .004 .128
	38 .011 .282	40 .007 .205	34 .009 .846	34 .001 .846	61 .003 .410	32 .002 .462	8 .003 .051	39 .005 .128	61 .003 .410	44 .004 .026	6 .004 .103	13 .003 .308
	13 .019 .154	26 .009 .077	4 .004 .026	7 .007 .077	18 .002 .103	13 .011 .308		12 .004 .205		17 .001 .026	49 .004 .051	66 .004 .333
	30 .011 .077	64 .005 .026		40 .001 .231	6 .004 .103	64 .002 .077		31 .001 .385		7 .002 .077	35 .004 .179	44 .002 .026
	48 .010 .026	63 .006 .077		11 .004 .179				6 .002 .103		34 .003 .846	13 .002 .308	4 .001 .026
	7 .005 .077	31 -.002 .333						36 .004 .282			55 .001 .026	35 .001 .179
		38 .005 .436										39 .005 .282
		23 .005 .231										
Cumulative correlation	.968	.982	.981	.984	.986	.987	.983	.987	.983	.987	.992	.992
Number of items	12/48	14/57	9/61	11/62	10/62	10/62	8/58	12/58	8/63	11/63	12/63	13/63

Table 9. Summarized SEQUIN Results
3/4 of Unique Activities/Project Within x · SD (Scored 1)

	Arithmetic						Percentage					
x · SD	0.1	0.25	1.0	1.25	1.50	2.0	0.1	0.25	1.0	1.25	1.50	2.0
Intercriterion correlation	0.015	-0.001	-0.147	-0.169	-0.185	-0.205	-0.137	-0.186	-0.210	-0.214	-0.206	-0.208
Evaluation mean	3.872	7.256	12.872	13.949	14.564	15.436	6.487	9.897	14.564	15.077	15.333	15.897
Selected activities	16 .811 .179	16 .777 .205	16 .875 .256	16 .880 .256	16 .873 .256	16 .871 .256	16 .828 .179	15 .808 .128	15 .829 .205	16 .834 .282	16 .839 .282	16 .833 .282
	33 .069 .051	63 .073 .051	47 .031 .103	47 .028 .103	18 .024 .103	26 .031 .231	38 .062 .128	11 .063 .128	16 .065 .256	15 .068 .231	26 .067 .231	15 .074 .231
	47 .029 .026	45 .030 .154	53 .030 .051	53 .041 .051	15 .036 .205	62 .048 .179	15 .035 .077	47 .025 .026	64 .048 .077	64 .030 .077	64 .029 .077	19 .032 .128
	41 .027 .026	18 .028 .051	61 -.000 .410	11 .003 .179	52 .013 .077	17 .015 .051	7 .019 .077	26 .013 .231	26 -.000 .231	52 .011 .077	52 .011 .077	6 .011 .103
	60 -.002 .051	40 .009 .077	22 .009 .256	61 .003 .410	61 .006 .410	19 -.003 .103	30 .004 .128	36 .035 .231	30 .016 .359	12 .004 .231	40 .004 .205	61 .013 .410
	23 .000 .128	52 .016 .051	63 .014 .128	39 .005 .282	13 .008 .308	40 .007 .231	53 .010 .151	12 .012 .205	39 .010 .282	40 .013 .205	12 .022 .231	30 .003 .385
	51 .002 .051	43 .010 .077	48 .004 .077	34 .004 .846	45 .006 .308	11 .005 .179	63 .007 .128	49 .007 .051	11 .006 .179	26 .013 .231	45 .009 .359	39 .007 .282
	25 .009 .077	19 .003 .026	62 .003 .179	6 .006 .077	12 .006 .205	52 .006 .077	12 .005 .128	61 .005 .308	44 .004 .026	45 .009 .359	15 .003 .231	11 .007 .179
	66 .007 .179	39 .007 .103	30 .007 .231	18 .006 .103	23 .006 .256	53 .005 .051	24 .001 .051	28 .002 .462	61 .002 .410	36 -.001 .308	8 -.000 .026	44 .003 .026
	5 .005 .051	46 .003 .026	8 .001 .026	26 .002 .179	6 .002 .103	34 -.001 .846	54 .002 .051	6 .005 .077	6 .001 .077	7 .005 .077	11 -.000 .179	26 .002 .231
	4 .005 .026	12 .003 .077	34 .004 .821	64 .003 .077	39 .002 .282	8 .001 .051	8 .002 .026	44 .002 .026	43 .001 .231	11 .002 .179	36 .004 .308	46 .003 .128
	29 .003 .128	38 .007 .410	15 .003 .179		32 .004 .359	59 .001 .026	19 .001 .051	16 .000 .205	28 .004 .513	34 .002 .846	47 .002 .077	
	13 .008 .154	7 .004 .077	17 .002 .026			61 .000 .410		46 .002 .051	21 .002 .359	6 .001 .077		
	40 .003 .026					15 .002 .205		33 .001 .051	46 .003 .128			
Cumulative correlation	.975	.970	.984	.981	.986	.987	.976	.982	.991	.989	.989	.990
Number of items	14/45	13/56	13/61	11/62	12/62	14/62	12/58	14/58	14/63	13/63	12/63	11/63

standard deviation. The "Intercriterion Correlation" is displayed representing the correlation between the internal variable (Evaluation Mean) and the external variable (inverse of the arithmetic difference or total estimated - total expended for the entire project). The "Evaluation Mean" is the mean of the number of unique activities per project with a score of one (1).

The items (activities) selected by SEQUIN as being the best predictors of the internal variable are contained in the upper left hand corner of the boxes which make up the body of the Tables 8 and 9. The internal variable is itself an indicator of estimating accuracy based on the criteria used for scoring. These same Item Numbers appear sequentially on the SEQUIN output and were selected down to a point where the correlation change became very small. The number directly below the Item Number is its Item Difficulty. Item Difficulty is a measure of how often an activity has a score of one (1) throughout the population of projects. The number in the upper right corner of the box represents the Correlation Change extracted from the SEQUIN output. This number represents the amount of predictive correlation (validity) that the activity contributes toward predicting the internal variable. The "Cumulative Correlation" is the sum of the correlation changes within the column. The row titled "Number of Items" simply represents the number of activities selected sequentially from the SEQUIN output to obtain the cumulative correlation noted. The second number is the total number of items identified by SEQUIN as having any predictive capability. Activities having an item difficulty less than 0.005 were excluded from consideration by SEQUIN (see Appendix 3, page 173 for items not considered in that sample run).

Table 10 summarizes from Tables 8 and 9 the number of occurrences of the most indicative activities selected by SEQUIN. Activities appearing less than five times were not included.

Table 10. Number of Occurrences of Most Predictive Activities

Item Number	1/2 Within x · SD		3/4 Within x · SD		Total	Occurs First
	Arith-metic	Per-centage	Arith-metic	Per-centage		
16	6	6	6	6	24	18
26	5	6	2	5	18	4
15	3	4	3	6	16	2
61	4	4	4	3	15	
39	0	6	3	2	11	
30	1	5	1	3	10	
6	1	2	2	4	9	
40	3	1	3	2	9	
11	1	0	2	5	8	
12	1	1	2	4	8	
13	3	3	2	0	8	
17	4	2	2	0	8	
34	3	1	3	1	8	
46	0	4	1	3	8	
64	3	1	1	3	8	
7	3	1	1	2	7	
45	1	2	2	2	7	
52	2	0	3	2	7	
18	1	2	3	0	6	
44	0	3	0	3	6	
53	2	0	3	1	6	
62	3	1	2	0	6	
8	0	1	2	2	5	
19	1	0	2	2	5	
47	0	0	3	2	5	
63	2	0	2	1	5	

Qualitative Analysis

The minimum of effort and planning accomplished in the Feasibility phase (B) is immediately apparent from Tables 4 and 5 (pp. 89 and 108) and Figures 5 and 6 (pp. 96 and 97). The mean and standard deviation of the differences for this phase (Figures 7 through 10, pp. 98-101) are extraordinarily variant as might be expected from the broad definitions of the activities and the nature of the work being accomplished. Control over this type work is rarely very strict since several minor tasks associated with the generalized activities are usually accomplished by several organizational entities outside of the development organization. Activity C010, Define System Concepts, is a generalized task closely related to this work and, as can be seen (Figure 9, p. 100), also has an extreme variance in estimating accuracy. Defining the problem carefully is critical to solving a problem. Therefore, the subsequent cyclic development, software reliability, testing demands, and schedule and cost overruns, may be directly related to the ill defined work, or lack thereof, in phase B [113, 90].

Estimating accuracy (Figures 9 and 10, pp. 100 and 101) can generally be said to be very bad and inconsistent. However, with a few exceptions, the activities in the Analysis phase (C) generally have smaller means and standard deviations for the analytical type of work being accomplished. Furthermore, this observation is supported by Figures 11 and 12 (pp. 102 and 103) "Hours underestimated versus hours overestimated." The number (17) of activities in the analysis phase contributes to a more precise definition of the work, making it easier to estimate and control projects.

The literature identifies underestimating as the predominant problem in estimating accuracy [133, 23, 72]. This study reflects, contrary to the literature, that overestimates are as common as underestimates. However, in several instances (e.g., critical activities C150, E010, E020, E070) the hours underestimated are more significant than the hours overestimated, while the number of activity

occurrences are equal or in an inverse proportion. This phenomenon is even more evident in the Testing phase (E) in contrast to all other phases (Figures 12 and 13, pp. 103 and 104). Clearly, purposely overestimating can obscure poor estimating where resource expenditures actually approach the high estimate. This could, of course, be argued as being accurate estimating. This might be possible, except for the predominance of overestimates in this data base, and the wild deviations in estimating accuracy. Gross overestimates are more likely to be accepted in government agencies as opposed to commercial activities. There is no competitive bidding for systems development among government agencies, and as yet, no one to seriously challenge the cost. Software development in the government has not yet experienced an Ernest A. Fitzgerald who achieved national attention from exposure of cost overruns on the C5-A military transport aircraft development. Furthermore, manpower authorizations in Air Force software development agencies are occasionally justified by the estimated workload figures for proposed systems development projects. Recall, however, that the Design Center develops standard systems used at many Air Force installations throughout the world. This multiplicity of system usage has considerable potential for amortizing the system cost and for magnifying savings and efficiencies.

The predominance of programming activities does not seem unusual (Figures 5, 6, 11, 12, 13, 14, pp. 96, 97, 102-105). As systems become better defined the work is more easily subdivided into discrete tasks (programs) each having its own set of activities. The literature implies that the programming phase is the most understood, disciplined and quantifiable [138]. Therefore, it is surprising that estimates for the activities of Coding (D020) and Program Test (D090) are so grossly inaccurate (Figures 9 and 10, pp. 99 and 100), as is much of the program development phase (D). A possible explanation for this observation is that the data system specifications for programs at the AFDSDC were never well written, if

written at all, during the early years of that organization's existence (1969-1974). Rarely did the specifications adhere to the standards prescribed for specification writing. Perhaps this flaw in professional software development can explain the estimating variance in these activities. Most writers, including Dijkstra, emphasize the critical importance of detailed specifications for predicting and controlling program development [106, 27, 97]. Requirement changes have been identified as the most common problem in the development cycle which have the most influence on schedule and cost overruns [17, 85]. The Design Center is no exception to the problem of changing requirements which may be another factor affecting estimating accuracy during program production. Accurate estimating is certainly much more difficult when the object to be estimated is moving so often, and in unanticipated directions.

There appears to be a reasonable distribution of skills across activities and phases (Figures 15 through 18, pp. 106 and 107). Only two exceptions are apparent:

1. the comparatively large amount of programmer hours expended (more than twice as many as any other skill, Figures 17 and 18, pages 106 and 107) [122], and
2. the almost insignificant amount of support hours portrayed in Figure 18 (p. 107).

Robert Barry suggests that valuable resources are wasted on the wrong part of the problem solution, i.e. coding [12]. Apparently the programmer skill is being excessively expended to compensate for inadequate analysis, incomplete specifications, lack of change control and inexperienced software development management [74]. The question posed earlier in this research suggested that if traditionally only 15% of the development time is spent on coding, then the productivity of programmers is not an issue deserving so much research. Nevertheless, a continuing and more detailed investigation of this disproportionate expenditure of programmer hours should be conducted. That is not to say that it is programmer

productivity per se which will resolve cost and estimating problems. The small amount of support hours displayed do not necessarily refute Pietrasanta's contention that support is a significant resource factor consistently ignored in making estimates [115]. First of all, the Implementation phase was eliminated from this research and a considerable amount of support effort is normally expended during that phase. The most important reason for small support figures is that the PARMIS system has never had an adequate procedure for collecting support hours expended, nor for automatically adding overhead according to some officially prescribed algorithm. The support hours portrayed are predominately for the typing and keypunch efforts of clerks.

An unusually apparent estimating problem is related to the gross overestimating for activities in phase G (Table 4, p. 89) related to developing and coordinating the user documentation (Figures 7 through 10, pp. 98-100). It is not clear what factors induce this situation but obviously these activities require closer attention when estimates are made. Perhaps the fact that the estimates for these activities are made early in the development cycle and the work is generally accomplished at the very end of the cycle contributes to the poor estimating record. Experience suggests that user documentation is sometimes done hurriedly and poorly, in less time than should be expended, so that deadline dates can be met.

The project data in Table 5 (p. 108) span the spectrum of small to large (new) software development projects (Figure 19, p. 115). The completeness of planning across the projects (i.e., the presence or absence of specific development phases) is somewhat inconsistent (Figures 20 through 23, pp. 116-119). Note, project numbers 22 and 23 have no Programming (phase D) activities but do consist of Testing activities (Table 5, p. 108). However, these are very small projects and may have involved only documentation changes in conjunction with the results of system testing. The lack of this kind of project definition information in PARMIS is also a handicap to research. More complete project information (i.e. type project, actual and estimated

start and completion dates, etc.) would be valuable in any research to analyze the impact of serious estimating errors on total project performance. Drawing strong conclusions regarding the projects would not be appropriate because in several cases, a considerable amount of data, in terms of activities, was ignored (Implementation phase) or discarded (non-standard activities).

Apparently, from Figures 20 through 23 (pp. 116-119), the percentage of effort expended by development phase, or by the "rule of thumb" classification is extremely inconsistent. But then the various "rules of thumb" themselves, noted in Chapter III, Literature Survey, are inconsistent, varying as much as 10%. This disparity in the distribution of effort is particularly obvious in terms of the number of activities in the five largest projects (Figures 19 through 23, pp. 115-119; projects 6, 9, 15, 20 and 30). Interestingly, but not very significant, the averages for the "rules of thumb" classifications fall within the range of values, as found in the literature:

1. Analysis @ 23%,
2. Programming @ 28%,
3. Testing @ 38%, and
4. Documentation @ 17%.

In the Design Center's classification of phases and activities (as opposed to the "rule of thumb" classifications) only the percentages for Programming (@ 46%) and Testing (@ 21%) change, and that is because individual program testing is accomplished in phase D (Figures 21 and 22, pp. 118 and 119).

Figure 24 (p. 120) reflects the emphasis of the programmer skill on these projects. Note, however, that some of the smaller projects (Figure 19, p. 115: projects 16, 17, 27, 28, 30, 31, 35, 37, 38) have a disproportionately high percentage of programmer use. Intuitively, this observation is consistent with how skills on small projects might be distributed (i.e., the programmer also becomes the analyst) (Figures 25 and 26, pp. 121 and 122).

A close examination of the individual projects in Tables 6 and 7 (pp. 123 and 125) reveals that the number of activities satisfying the criteria (scored 1) improves significantly when the difference (arithmetic or percent) is within one standard deviation. This is not unexpected since the same data used to calculate the standard deviation were used in the scoring. Thereafter, up to twice the standard deviation (2 · SD), the number of activities scored does not improve readily. This point is more apparent in the Average Number of Activities plus Phases scored per project (Evaluation Mean) for each criterion, noted at the end of both tables. Thus, on the average, between 70% and 83%, of one half and three fourths of the unique activities in a project, fall within one standard deviation of the arithmetic and percentage differences. The above range of percentages was obtained by dividing the smallest and largest average results (12.879 and 15.205) from Tables 6 and 7 (pp. 123 and 125) by the average number of unique activities (plus phases) per project (18.282).

According to the figures in Tables 8 and 9 (pp. 127 and 128) there is no indicative correlation between the internal and external variables. The number of Selected Activities required to attain a minimum of 97% of the predictive capability associated with specific activities, in any one run, never exceeds fourteen (14) items and is frequently less. Selected Activities are generally consistent across the 24 different tests. This observation is apparent from examining Tables 8 and 9, and even more obvious from the summarized data in Table 10 (p. 130). The first four items (16, 26, 15, 61) in Table 10 can be said to be good indicators of estimating accuracy. That is, if the difference between the estimated and expended hours of most (50% to 75%) of the specific activities in a project are within one standard deviation of the differences for these activities, then it can be anticipated that 70% to 83% of most of the unique activities in the project will also be within one standard deviation. It is equally interesting to note that there is little change in the

most predictive activities regardless of whether the arithmetic or percentage standard deviations were used for scoring.

The single most predictive activity is Item Number 16, identified from Table 11 as activity C070, Defining Input Data (Media and Volume). The complete definition for this activity is contained in Appendix 2 and should be examined carefully. There is an intuitive appreciation for the conspicuous emergence of such an activity. This is the work which has the most to do with how well the remainder of a software development effort will proceed in terms of estimating accuracy. The importance of carefully defining the system input, as well as accurately estimating this effort, is further substantiated by the frequent occurrence of Item 13 (activity C040, Identify Data Base Requirements/Data Elements/Codes) as a prominent indicator. Finally, this research confirms emphatically and quantitatively what software development practitioners have long known: that inadequate estimates of the work required for detailed system definition will usually render inaccurate the remaining manpower cost and schedule estimates. This contention is supported by the prominence of Items 16, 15, 11, 12 and 13 in Table 10 (p. 130).

The emergence of Item Number 61 (activity G120) is not unusual in light of the previous observations (Figures 7 and 11, pp. 98 and 102) of estimating accuracy within the User Documentation phase (G). Gross overestimates, of this activity in particular, create an extremely large standard deviation within which most of the occurrences of activity difference would fall. Thus this activity would be scored one (1) more often than not. Since the activity occurs late in the development cycle it is not of much value in terms of identifying estimating problems early enough for management actions to be greatly effective. Furthermore, the range of the standard deviation creates some apprehension in using the estimating accuracy on this activity as an indicator for even the remainder of the latter stages of development. Nevertheless, the conspicuousness of this activity, in conjunction with the previous observations, should act as a clear

warning signal to management regarding estimates in this important part of software development.

The last six (6) Item Numbers (67-72) appearing in the sample SEQUIN output corresponded to the gross activities used in "small" projects (K100 through K500 plus Summary phase). No data were processed for projects using these activities and therefore the Item Numbers always received a zero (0) score and had no effect on the final results. The summarized information pertaining to these activities was retained in Table 4 (p. 89).

Table 11. Item Number - Activity Identification

Item Number	Phase/ Activity Code	Definition
1	B100	Research Problem Requirement
2	B110	Economic Analysis
3	B200	Prepare Data Automation Request (DAR)
4	B210	Review/Coordinate/Approve DAR
5	B300	Prepare Data Project Directive (DPD)
6	B310	Review/Coordinate/Approve DPD
7	B400	Prepare Data Project Plan (DPP)
8	B410	Review/Coordinate/Approve DPP
9	B000	Summary Phase B
10	C010	Define System Concepts
11	C020	Define Interface/Integration Requirements
12	C030	Review/Coordinate Interface/Integration Requirements
13	C040	Identify Data Base Requirements/Data Elements/ Codes
14	C050	Define Detailed System Requirements and Objectives
15	C060	Flow Chart System Processes
16	C070	Define Input Data (Media and Volume)
17	C080	Identify Timing Factors (simulation)
18	C090	Analyze System Optimization
19	C100	Define Audit Trail Requirements
20	C110	Coordinate System With User
21	C120	Document System Specifications
22	C130	Define Detailed System Processes
23	C140	Prepare Detailed System Flow Charts
24	C150	Prepare and Review Program Specifications
25	C160	Prepare System Test Data
26	C161	System Design Review
27	C000	Summary Phase C
28	D010	Program Analysis and Flow Charting
29	D020	Code
30	D030	Keypunch
31	D040	Compile/Assemble
32	D050	Desk Check/Debug
33	D071	Prepare Test Data
34	D090	Test Program
35	D120	Test Program in Subsystem
36	D130	Prepare Program Folder (Vol. IV Documentation)
37	D000	Summary Phase D

Table 11. (continued)

Item Number	Phase/ Activity Code	Definition
38	E010	Test System
39	E020	Debug System
40	E050	User Review/Coordinate/Approve
41	E070	Operational Test/User Approval
42	E080	Prepare System for Release to Operations Directorate
43	E000	Summary Phase E
44	F500	Announce Documentation Release in USAF Publication
45	F520	Prepare Vol. I Documentation-Operations Manual (Run Book)
46	F530	Type Vol. I Draft
47	F540	Coordinate Vol. I with User
48	F550	Prepare Vol. I for Quality Control and Environmental Test
49	F570	Correct Vol. I
50	F620	Prepare Vol. II Documentation - Program Documentation Specifications, etc.
51	F630	Type Draft
52	F640	Coordinate
53	F650	Prepare for Quality Control and Environmental Test
54	F670	Correct
55	F720	Prepare Documentation - Unique for Batch Type Systems and Computers
56	F730	Type
57	F740	Coordinate
58	F750	Prepare for QC
59	F770	Correct
60	F000	Summary Phase F
61	G120	Prepare Vol. III Documentation - Functional Users Manual
62	G130	Type Draft
63	G140	Coordinate
64	G150	Prepare for QC
65	G170	Correct
66	G000	Summary Phase G

Table 11. (continued)

Item Number	Phase/ Activity Code	Definition
67	K100	Feasibility - Small Projects
68	K200	Analysis
69	K300	Programming
70	K400	Test
71	K500	Documentation
72	K000	Summary Phase K

CHAPTER VII

CONCLUSIONS/RECOMMENDATIONS

The Engineering Approach

Change is the predominant characteristic of the computer industry [108]. The people associated with the computer science discipline are generally classified with engineers because the computer itself is the most predominant symbol of "the industry" and because it is recognized as an engineering marvel. On the other hand, at the 1969 Conference sponsored by NATO, entitled "Software Engineering," it was admitted that:

> The phrase "Software Engineering" was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines that are traditional in the established branches of engineering. [106]

Hardware and software are constantly achieving remarkable strides in making the power of the computer more beneficial to mankind and more easily available. There is, however, a significant departure between hardware and software in terms of how this progress is achieved. Hardware progresses in a scientific manner through research and development and the application of engineering principles. Software development progresses by brute force and trial and error, and its development generally proceeds in an undisciplined manner. In short, the software development process has fallen far behind the scientific approach used for hardware development and may restrain the effective use of hardware capabilities in the future [66, 132].

A New Development Model

The foremost conclusion from this research is that estimation of software development continues to be as bad as, or worse than, it has ever been. The process used for software development is incompatible

with the classical estimating techniques which managers attempt to superimpose on this process [97]. That is not to say that there are not standard activities which must be estimated and accomplished in some sequential order to successfully develop and implement software. The structure of this process is faulty when combined with the estimating objective of trying to predict the cost and span time for the entire project development [30]. These observations deserve further elaboration.

Primarily, the process of software development should be restructured into a new Research - Development - Production (R-D-P) Model. The R-D-P environment is used by industry and government in the development of new products, advanced technology (NASA), or a new weapon system (C5-A aircraft) [27]. New software development is no less complex than any of these endeavors. In both R&D and software development, heavy expenditures of resources are required in order to produce future benefits that are very uncertain in their magnitude. Because of long software project lives, attempts at total project planning and cost estimation are very inaccurate. The research aspect of software development also significantly affects the accuracy of software project estimates.

Thayer believes that it is impossible to analyze and put into writing what a man does to manage a Command and Control System. For example, despite some $65 million spent in writing the software for the Strategic Air Command's 465L Command and Control System, 95% of the system had to be rewritten due to poor initial analysis and hastily conceived assumptions [146]. The results of this dissertation once again confirmed that the lack of preparation and definition of work during Feasibility Studies and Requirements Analysis (preliminary system design) results in the wild variance of estimates for these efforts. The activities in these two phases should be classified as Research in lieu of the classical phase structure which has predominated--and which has consistently failed as a framework for accurate estimating. The Development of software should

encompass the detailed system design (specifications), program development, testing and documentation. Production should involve those activities required in Implementation, Evaluation and Modification. Maintenance of software is a continuing process and should remain separately funded and controlled apart from the evolution of the original system. It is not critical at this point to be overly concerned about which specific activities belong in which R-D-P phase. These classifications will change with experience and there will always be interaction between even these new phases, and some iteration within the total process [54].

While it is doubtful that the software development process can ever be delineated into clear cut phases with precise cut off points, the estimation of resources and time requirements for the phases of R-D-P can be restricted. Hopefully, this R-D-P approach will heighten managements' awareness of the complexity of what has to be done and eliminate the historically impossible requirement to accurately estimate the cost and time for the entire software development project. Furthermore, this approach would restrain the rush to begin coding [13].

The Research phase should be more finitely defined in terms of standard activities. In fact, the National Bureau of Standards should be as concerned with standardizing this process as it has been with standardizing computer languages. Adequate planning and report flexibility can be easily incorporated with the standard activities, as is done in PARMIS, to satisfy individual managers and existing commercial project control systems. The Research phase should be estimated independently and the estimates for the subsequent Development and Production should not be attempted until the Research phase has been brought to a mutually satisfactory conclusion. Precise objectives and end products should be specified for most activities and certainly for each phase. These new classifications of effort provide realistic management decision points and a basis for more accurately estimating the next phase.

Unfortunately, these recommendations are certainly not revolutionary ideas. They have been proposed by other, more renowned, authors since 1963 [88, 13, 106]. It is inconsistent for an industry so dependent on change to be so resistant to change. This resistance is particularly confounding in an area of the science (i.e., software engineering) which has proven time and again that its primary function (reliable software development) cannot be controlled in terms of cost and time. Quality research and good management practices are not incompatible.

Supporting the Hypothesis

The findings of this research support the original hypothesis. Specific resource consuming activities have been isolated which are consistently good indicators of whether a software development project is accurately estimated. These activities were identified as a result of quantitative analysis. Furthermore, the isolated activities have an intuitive and logical appeal since they are predominately concerned with initial efforts at defining the system in terms of the data elements and files to be developed. The accuracy of the original estimates in defining the effort of these complex tasks obviously has a direct impact on the accuracy of the subsequent resource estimates for the remaining activities in software development. Considerable software development experience has been cited throughout this study confirming that when these types of analysis/design activities must be reaccomplished, large schedule and cost overruns result. Software development management is therefore liable to be knowledgeable of, and to concentrate their management attention and efforts on these important predictor activities. Original estimates for these indicative activities should be given careful consideration. Subsequent software development planning should obviously be updated in light of the results of the estimating accuracy on these illuminating tasks. Fortunately these indicative activities occur early enough in the development cycle

where honest admissions of difficulty still leave sufficient opportunities for various management actions (i.e., reduced objectives, etc.) which could save the schedule, the project, the contract and/or personnel morale.

Chapter I, Introduction, posed a question concerning a correlation between specific activity estimating accuracy and the final project estimated and expenditure totals. SEQUIN, using these data, could find no correlation between its Objective Function and the external variable used (the inverse of the final arithmetic difference). Clearly some relationship must exist between accurately estimating individual activities and the difference between the project's final total estimate and expenditure. It is conceivable that either the predominance or overestimates in this data source obscured such a correlation or that an appropriate external variable was not identified in the data base.

Centralized Control

Worthy of additional emphasis is the problem associated with the limitations of data sources dealing with estimating the software development process. Data are now available in numerous management information systems. The data elements used, the development process, and definitions of the individual activities in the software development environment, pertinent to project planning, progress, and change, must be standardized. It is recommended that a standardized, PARMIS like, system be centrally established for controlling most software development done by and for the U.S. Government. A precedent for this type of centralized function was established with the evolution of the Federal Simulation Laboratory (FEDSIM) in Washington, D.C. This organization, under operational control of the U.S. Air Force, conducts simulations of automated computer systems for all Federal agencies. Clearly, this approach would reduce the costs of the many project control systems, manual or automated, now operated by the various government agencies

involved in software development. Input could be mailed or dispatched over government data networks. On-line access could be provided for immediate status inquiries. Standard management reports would not be required more than semi-monthly; therefore response time should not be an insurmountable problem. An important consideration would be to limit the system to project status reporting and accumulation of valuable historical data as opposed to man-hour accounting. Mathematical tools and management analysis techniques could be used to continually analyze and refine the historical data. Significant planning factors might be isolated and time and cost estimating guides could be identified and fed back to the agencies. At least some continued monitoring of the estimating variability of specific activities would tend to narrow the difference between estimates and expenditures. A repository for sharing software development experience, both good and bad, could be established. Standard software evaluation techniques could be developed for use at various development stages as opposed to waiting until the system test for such evaluations. The possibilities for all kinds of improvements in research, software development, project control, cost visibility, estimating accuracy, reliability and management are numerous. Research and development of the software development process itself would certainly be enhanced [78].

> The rapid strides of professional progress come when the structure and principles that integrate individual experiences can be identified and taught explicitly rather than by indirection and diffusion. The student can then inherit an intellectual legacy from the past and build his own experience upward from that level, rather than having to start over again at the point where his predecessors began. [72]

Recommendations

The remaining recommendations require no elaborate discussion and need only be listed.

1. Planning is crucial to estimating accuracy. As demonstrated by Adams in his dissertation [2], a professional conscientious effort

must be made to consider all resource consuming activities and to make some estimate of the magnitude of the effort regardless of the amount of information available.

2. Dynamic planning is essential. Plans and estimates must be realistically updated as more accurate information becomes available in conjunction with periodic project status reviews. Update planning is a management function--not a management failure! Pushing people to attain unrealistic goals destroys morale, adversely influences system reliability and encourages false progress reporting. Product, design and status reviews should be planned activities.

3. Improve reporting of productive work. Better, more encompassing methods for capturing all resource expenditures associated with a specific software development project are required.

4. Measurement of software development progress should not be limited to a single variable. Several different indicators of progress should be defined:

(a) dates,
(b) hours,
(c) cost,
(d) skill usage,
(e) support contributions,
(f) numbers of program tests,
(g) approved completed products such as specifications, programs and chapters,
(h) number of design changes, and
(i) number and magnitude of estimate changes, etc.

Effective management visibility requires that the Research or Development efforts be amenable to review at anytime with little or no preparatory cost [97].

5. New data elements are required. Project control systems need additional information to enhance interpretations of project progress, to aid in project control, and for subsequent research. Such data should at least include validated reasons for changes to plans and estimates and short definitions of the project objectives.

6. The data system specifications are the most important product generated during the software development process. The term "software development" encompasses the specification of all procedures to be accomplished by both man and machine. McHenry [97] has postulated an alternative software development model which is based on the recognition of programming as a specifying activity. The postulated process is a specification continuum. The process concludes when the specifications satisfy a test procedure by execution. The postulation has as its justification the continuous potential for managerial testing, measuring, reviewing and controlling. The R-D-P Model proposed by this dissertation and McHenry's Specification Model are totally compatible and complementary.

7. The development of user documentation has an unusually significant influence on software development estimating accuracy. This observation, combined with the history of grossly overestimating this activity, warrants a recommendation that more management attention be given to estimating this phase of development.

8. New techniques for organization and technical software development such as Chief Programmer Teams and structured programming should continue to be exploited and enhanced.

9. Activity and project estimates should be classified according to some scheme which evaluates the reliability (risk, confidence) of the estimate.

10. Software estimating review groups should be established similar to those used at TRW and described by Wolverton [158]. Tracking the estimating accuracy of these groups as well as determining the reasons for poor estimates, must, in the long run, improve estimating performance.

11. The various techniques for developing more reliable software, and controls to insure a reasonably accurate range for the time and cost variables, need to be interrelated and combined into a total organization control system concept. Software managers cannot, however, rely completely on new system analysis, programming

and testing technologies, organizational innovations and control systems. The primary factor in the total software development management equation is, and will always be, people. Recommendations on how to best manage the people factor in software development is a most complex issue which even Weinberg [151] has only begun to address.

12. Certainly additional studies can now be initiated using similar techniques, with other data sources, to isolate the activities pertinent to those sources which significantly influence software development project estimates. Additional important correlations may be discovered between specific activities and overall project success in terms of estimating accuracy. The excessive expenditure of programmer resources should be examined in greater detail. There is still much to discover about the software development process, and, as with all other basic research, that knowledge will not all be uncovered by any single study. At least applying the knowledge and recommendations gained from this research might well yield a marked improvement in future software development project estimating efforts.

Summary

Peter Drucker has succinctly described the problem of the transition of the data automation profession into a controlled development cycle.

> To make knowledge work productive will be the management task of this century, just as to make manual work productive was the great management task of the last century. The gap between knowledge work that is managed for productivity and knowledge work that is left unmanaged is probably a great deal wider than was the tremendous difference between manual work before and after the introduction of scienfitic management. [47]

Nevertheless, the transition must be made. The sharing of experimental control systems data, needed research to discover better ways of planning and estimating the software development cycle, and the advent of a more knowledgeable software project manager are some of the possible solutions which must be pursued. This is

dependent of course on whether the "software engineer" wants to control the software development process, or whether the process will continue to control him.

REFERENCES

1. Abernathy, W. J. Subjective estimates and scheduling decisions. Management Science 18, 2 (Oct. 1971), 80-87.

2. Adams, J. R. The estimation process: Network analysis and factors affecting accuracy. Ph.D. Th., Syracuse University, Syracuse, N.Y., Feb. 1974.

3. Alderton, A. D. Getting a measure. Data Processing 12, 6 (Nov.-Dec. 1970), 470-477.

4. Amory, W., and Clapp, J.A. Engineering of quality software systems: A software error classification methodology. Tech. Rep. AD A007 772, Defense Documentation Center, Alexandria, Va., Jan. 1975.

5. Aron, J. D. Estimating resources for large programming systems. IBM Tech. Rep. FSC 69-5013, Federal Systems Div., Gaithersburg, Md., 1969.

6. Aron, J. D. Programming development techniques overview. IBM Tech. Rep. FSC 71-6021, Federal System Div., Gaithersburg, Md., 1971.

7. Auxco Inc. Project Management System. A commercial introductory manual. Auxco Computer Enterprises Inc., New York, N.Y., Nov. 1972.

8. Avots, I. Why does project management fail? Management Review 59, 1 (Jan. 1970), 36-41.

9. Axsmith, D. J. Planning and control for computer systems projects. in Data Processing XIII, DPMA, Park Ridge Ill., 1968, pp. 121-138.

10. Baker, F., and Mills, H. D. Chief programmer teams. Datamation 19, 12 (Dec. 1973), 58-61.

11. Baker, F. T. Structured programming in a production programming environment. Proc. of 1975 International Conf. on Reliable Software, Los Angeles, Cal., Apr. 1975, pp. 172-183.

12. Barry, B. (Ed.) Software lifecycle management techniques. Tech. Rep. AD 008 369, Defense Documentation Center, Alexandria, Va., Oct. 1974.

13. Bauer, F. L. (Ed.) Advanced Course on Software Engineering. Springer-Verlag, Berlin, 1973, pp. 522-545.

14. Baumgartner, J. S. Project Management. Richard D. Irwin Inc., Homewood, Ill., 1963.

15. Bemer, R. W. (Ed.) Computers and crisis: how computers are shaping our future. Proc. of ACM National Conf., New York, Sept. 1970, pp. 288-296.

16. Benjamin, R. I. Control of the Information System Development Cycle. Wiley-Interscience, New York, 1971.

17. Boehm, B. W. Software and its impact. Datamation 19, 5 (May 1973), 48-59.

18. Bosh, C. A., and Boehm, B. W. Software development characteristics CCIP-85 study group. TRW Systems Corp., Redondo Beach, Cal., 1971.

19. Brandon, D. H. The Economics of Computer Programming. in On the Management of Computer Programming. G. L. Weinwurm (Ed.) Auerbach Inc., Princeton, N.J., 1970, pp. 3-16.

20. Bratman, H., and Court, T. The software factory. Computer 8, 5 (May 1975), 28-37.

21. Brown, C. L. Is control a dirty word? Management Review 61, 8 (Aug. 1972), 56-58.

22. Brooks, F. P. Why is software late? Data Management 9, 8 (Aug. 1971), 18-21.

23. Brooks, F. P. The Mythical Man-Month. Addison-Wesley, Reading, Mass., 1975.

24. Bullen, R. Engineering of quality software systems: software first concepts. Tech. Rep. AD A007 768, Defense Documentation Center, Alexandria, Va., Jan. 1975.

25. Buxton, J. H., and Randell, B. (Eds.) Software Engineering Techniques. NATO Science Committee, Brussels, Apr. 1970.

26. Cammack, W. B., and Rodgers, H. J. Improving the programming process. IBM Tech. Rep. TR 00.2483, Systems Development Div. Poughkeepsie, N.Y., Oct. 1973.

27. Canning, R. Managing the programming effort. EDP Analyzer 6, 6 (June 1968), 1-15.

28. Carter, D. M., Gibson, H. L., and Rademacher, R. A. A study of critical factors in management information systems for the U.S. Air Force. Tech Rep. AD 777 301, Defense Documentation Center, Alexandria, Va., Oct. 1973.

29. Cheng, L. L. Engineering of quality software systems: some case studies in structured programming. Tech. Rep. AD A007 771, Defense Documentation Center, Alexandria, Va., Jan. 1975.

30. Clapp, J. A. Software engineering--problems and future developments. Tech. Rep. AD A003 422, Defense Documentation Center, Alexandria, Va., Nov. 1974.

31. Clapp, J. A., and LaPadula, L. J. Engineering of quality software systems. Tech. Rep. AD A007 766, Defense Documentation Center, Alexandria, Va., Jan. 1975.

32. Clark, R. N., Nelson, E. A., and Petersen, K. E. Data automation resource-reporting and management system. Tech. Memo. TM-3558/000/01 System Development Corp., Santa Monica, Cal., Aug. 1967.

33. Conforti, J. J. Project estimation can be easy. Journal of Systems Management 23, 11 (Nov. 1972), 36-39.

34. Conway, M. E. How do committees invent? Datamation 14, 4 (Apr. 1968), 28-31.

35. Cosgrove, J. Needed: a new planning framework. Datamation 17, 23 (Dec. 1971), 37-39.

36. Couger, J. D. Evolution of business system analysis techniques. Computing Surveys 5, 3 (Sept. 1973), 167-198.

37. Craig, G. R. Software reliability study. Tech. Rep. AD 787 784, Defense Documentation Center, Alexandria, Va., Oct. 1974.

38. David, I., and Ditri, A. E. Planning of data systems. Data Processing XIII, DPMA, Park Ridge, Ill., 1968, pp. 177-189.

39. Definite milestones essential for software development. Management Advisor, 9, 8 (July 1972), 3.

40. Delaney, W. A. Predicting the cost of computer programs. Data Processing Magazine, 8, 10 (Oct. 1966), 32-34.

41. Dennis, J. B. The design and construction of software systems. in Advanced Course on Software Engineering. F. L. Bauer (Ed.) Springer-Verlag, Berlin, 1973, pp. 1-27.

42. Dickson, G. W. Control systems for information systems development projects. AEDS Journal 4, 1 (Sept. 1970), 29-34.

43. DiCola, J. N. Scheduling creative resources. Management Review 59, 9 (Sept. 1970), 55-60.

44. Dienemann, P. Estimating cost uncertainty using Monte Carlo techniques. Tech. Rep. AD 629 082, Defense Documentation Center, Alexandria, Va., Jan. 1966.

45. Dooner, N. P., and Lourie, J. R. The application software engineering tool. IBM Research Rep. RC5434, Thomas J. Watson Research Center, Yorktown Heights, N.Y., May 1975.

46. Douglas, A. S. The conference on the planning and control of large programming projects. Computer Bulletin 15, 1 (Jan. 1971), 39-40.

47. Drucker, P. The Age of Discontinuity. Harper and Row, New York, 1969.

48. Eddington, A., The Nature of the Physical World, University of Michigan Press, Ann Arbor, Mich., 1958.

49. Farquhar, J. A. A preliminary inquiry into the software estimation process. Tech. Rep. AD 712 052, Defense Documentation Center, Alexandria, Va., Aug. 1970.

50. Farr, L., and Nanus, B., Factors that affect the cost of computer programming. Tech. Rep. AD 477 329, Defense Documentation Center, Alexandria, Va., June 1964.

51. Farr, L., and Zagorski, H. J. Factors that affect the cost of computer programming, Vol. II, a quantitative analysis. Tech. Rep. AD 607 546, Defense Documentation Center, Alexandria, Va., Sept. 1964.

52. Farr, L., LaBolle, V., and Willmorth, N. E. Planning guide for computer program development. Tech. Rep. AD 734 358, Defense Documentation Center, Alexandria, Va., May 1965.

53. Fisher, P., Hankley, W., and McCall, J. (Eds.) Steps toward reliable software: proceedings of a workshop. Tech. Rep. AD A010 396, Defense Documentation Center, Alexandria, Va., Jan. 1975.

54. Fleischer, R. J. Engineering of quality software systems: effect of management philosophy on software production. Tech. Rep. AD A007 767, Defense Documentation Center, Alexandria, Va., Jan. 1975.

55. Fleishman, T. Current results from the analysis of cost data for computer programming. Tech. Rep. AD 637 801, Defense Documentation Center, Alexandria, Va., June 1966.

56. Gayle, J. B. Multiple regression techniques for estimating computer programming costs. Journal of Systems Management 22, 2 (Feb. 1971), 13-17.

57. Gilb, T. Datametrics. Data Management 10, 10 (Oct. 1972), 35-36.

58. Gilb, T. Software metrics. Data Management 13, 7 (July 1975), 34-37.

59. Gildersleeve, T. R. The time estimating myth. Datamation 19, 1 (Jan. 1973), 47-48.

60. Goff, N. Development project costs. Journal of Systems Management 25, 9 (Sept. 1974), 14-16.

61. Gradwohl, A. J., and Wootan, W. O. Phase II final report on use of Air Force ADP experience to assist Air Force ADP management, Vol. III. Tech. Rep. AD 646 868, Defense Documentation Center, Alexandria, Va., Dec. 1966.

62. Gradwohl, A. J. Air Force ADP experience handbook (pilot version). Tech. Rep. ESD-TR-66-672, Electronic Systems Div., Hanscom Field, Mass., Dec. 1966.

63. Hatter, R. J. CCIP study regarding analysis of TRW software analysis data (excerpt). Lulejian and Assoc. Inc., Redondo Beach, Cal., 1971.

64. Herschauer, J. C., and Nabielski, G. Estimating activity times. Journal of Systems Management 23, 9 (Sept. 1972), 17-21.

65. Hill, P. B. The control of large scale software projects. IAG Journal 3, 4 (Dec. 1970), 394-405.

66. Hoare, C. A. R. Computer Programming as an engineering discipline. Electronics and Power 19, 8 (Aug. 1973), 316-330.

67. Horowitz, E. (Ed.) Practical Strategies for Developing Large Software Systems. Addison-Wesley, Reading, Mass., 1975.

68. Hurtado, C. D. Quantitative tool for decision making. Data Processing Management 9, 3 (Mar. 1971), 19-22.

69. International Systems Inc. Project Analysis and Control. A commercial introductory manual. International Systems Inc., King of Prussia, Penn., 1969.

70. Jackson, B. J. Evaluating programmer workload. The Australian Computer Journal 2, 1 (Feb. 1970), 22-26.

71. Jones, M. M., and McLean, E. R. Management problems in large scale software development projects. Industrial Management Review 11, 9 (Sept. 1970), 1-15.

72. Jones, M. V. System cost analysis: A management tool for decision making. Tech. Rep. AD 624 893, Defense Documentation Center, Alexandria, Va., Nov. 1965.

73. Jones, M. V. Estimating methods and data sources used in costing military systems. Tech. Rep. AD 626 153, Defense Documentation Center, Alexandria, Va., Dec. 1965.

74. Kay, R. H. The management and organization of large scale software development projects. Proc. AFIPS SJCC Vol. 34 AFIPS Press, Montvale, N.J., 1969, pp. 425-433.

75. Kelly, J. F. Cost estimation becomes less of an art--and more of a science. Production 58, 6 (Dec. 1966), 174-177.

76. King, W. R., and Wilson, T. A. Subjective time estimates in critical path planning--a preliminary analysis. Management Science 13, 5 (Jan. 1967), 307-320.

77. Koontz, H., and O'Donnell, C. Principles of Management. McGraw-Hill, New York, 1964.

78. Kosy, D. W. Air Force command and control information processing in the 1980's. Tech. Rep. AD A017 128, Defense Documentation Center, Alexandria, Va., June 1974.

79. Krauss, L. I. Administering and Controlling the Company Data Processing Function. Prentice-Hall, Englewood Cliffs, N.J., 1969.

80. Kriebel, C. H. The evolution of management information systems. in Operation Research in the Design of Electronic Data Systems, F. Haussman (Ed.), Proc. of NATO Conference in Munich, The English Universities Press, London, 1971, pp. 7-18.

81. LaBolle, V. Estimation of computer programming costs. Tech. Rep. AD 450 779, Defense Documentation Center, Alexandria, Va., Sept. 1964.

82. LaBolle, V. Development of equations for estimating the cost of computer program production. Tech. Rep. AD 637 760, Defense Documentation Center, Alexandria, Va., Apr. 1966.

83. Laschenski, J. P. Three tools for the management of a programming project. IBM Tech. Rep. TR 00.2111, Systems Development Div., Poughkeepsie, N.Y., Oct. 1970.

84. Lecht, C. P. The Management of Computer Programming Projects. American Management Assoc. Inc., New York, 1967.

85. Liquori, F., Software management through specification control. Software Age 2, 5 (May 1968), 8-34.

86. Littrell, R. F. A step toward quality control in computer programming: understanding the psychology of the management of computer programmers. Proc. ACM National Conf. ACM New York, N.Y., 1973, pp. 419-423.

87. Lombaers, H. J. (Ed.) Project Planning by Network Analysis. North Holland Publishing Co., Amsterdam, 1969.

88. Maher, J. J. (Ed.) Proceedings of the symposium on managing the development of large computer program systems. Tech. Rep. AD 652 982, Defense Documentation Center, Alexandria, Va., Jan. 1963.

89. Management Planning Guide for a Manual of Data Processing Standards. IBM Manual C20-1670-2, Mar. 1972, pp. 14-20.

90. Manley, J. H., and Archibald, R. W. (Eds.) Proceedings of the aeronautical systems software workshop. Tech. Rep. AD A004 547, Defense Documentation Center, Alexandria Va., July 1974.

91. Margolis, M. A. CAIG: In pursuit of improved cost estimates. Defense Management Journal 11, 1 (Jan. 1975), 20-24.

92. Martin Marietta Corp. PLANS: A Programming System for Developing Software: Related to Scheduling, Resource Allocation, and Project Control. Commercial introductory material. Martin Marietta Corp., Denver Colo., undated.

93. McConkey, D. D. 20 ways to kill management by objective. Management Review 61, 10 (Oct. 1972), 4-13.

94. McFarlan, F. W. Problems in planning the information system. Harvard Business Review 49, 2 (Mar.-Apr. 1971), 75-89.

95. McGauley, L. E. Can software adopt hardware controls? Data Management 9, 10 (Oct. 1971), 38-41, 62.

96. McGregor, D. On the Human Side of Enterprise. McGraw-Hill, New York, 1960.

97. McHenry, R. C. Thoughts on improving the software development process. IBM Tech. Rep. FSC 73-0330, Federal Systems Div., Gaithersburg, Md., June 1973.

98. McKeever, J. M., and Kruse, B. Management Reporting Systems. Wiley-Interscience, New York, 1971.

99. Metzger, P. Managing A Programming Project. Prentice-Hall, Englewood Cliffs, N.J., 1973.

100. Miller, E., and Lindamood, G. Structured programming: top down approach. Datamation 19, 12 (Dec. 1973), 55-57.

101. Morin, L. H. Estimation of resources for computer programing projects. Masters Th., University of North Carolina, Chapel Hill, N.C., 1974.

102. Morse, R. V. Control systems for better project management. Computer Decisions 3, 7 (July 1971), 28-31.

103. Murphy, E. L. Statistical methods of measuring the uncertainty of cost estimates. Tech. Rep. AD 718 862, Defense Documentation Center, Alexandria, Va., Feb. 1970.

104. Myers, G. J. Estimating the costs of a programming system development project. IBM Tech. Rep. TR 00.2316, System Development Div., Poughkeepsie, N.Y., May 1972.

105. Nashman, A. E. Software development management: the key to quality software products. Electronics and Aerospace Systems Convention, EASCON 74 Record, IEEE, New York, 1974, pp. 31-35.

106. Naur, P., and Randell, B. (Eds.) Software Engineering, NATO Science Committee, Brussels, 1969.

107. Nelson, E. A. Research into the management of computer programming: some characteristics of programming cost data from government and industry. Tech. Rep. AD 642 304, Defense Documentation Center, Alexandria, Va., Nov. 1965.

108. Nelson, E. A. Management handbook for the estimation of computer programming costs. Tech. Rep. AD 648 750, Defense Documentation Center, Alexandria, Va., Mar. 1967.

109. Nelson, E. A. Cost reporting for development of information processing systems. Tech. Rep. AD 657 793, Defense Documentation Center, Alexandria, Va., Apr. 1967.

110. Nelson, E. A. Some recent contributions to computer programming management. Tech. Rep. AD 680 105, Defense Documentation Center, Alexandria, Va., Apr. 1968.

111. Nelson, E. A. Managing the economics of computer programming: current methodological research. Proc. of ACM National Conf. Brandon/Systems Press, Princeton, N.J., Aug. 1968. pp. 346-349.

112. Nelson, E. A. Methods of obtaining estimates of computer programming costs: a taxonomy. Tech. Rep. AD 665 478, Defense Documentation Center, Alexandria, Va., Aug. 1968.

113. Parnas, D. Some conclusions from an experiment in software engineering techniques. Tech. Rep. AD 759 249, Defense Documentation Center, Alexandria, Va., June 1972.

114. Parsons, J. A. Matrix methods--decision making under uncertainty. Journal of Systems Management 23, 8 (Aug. 1972), 43-44.

115. Pietrasanta, A. M. Managing the economics of computer programming: current methodological research. Proc. of ACM National Conf. Brandon/Systems Press, Princeton, N.J., Aug. 1968, pp. 341-346.

116. Pietrasanta, A. M. Resource analysis of computer program system development. in On the Management of Computer Programming. G. F. Weinwurm (Ed.), Auerbach, Inc., Princeton, N.J., 1970, pp. 67-87.

117. Pietrasanta, A. M. Functional estimating of computer program system development. in On the Management of Computer Programming. G. F. Weinwurm (Ed.), Auerbach, Inc., Princeton, N.J., 1970, pp. 89-107.

118. Pooch, U. W. SEQUIN: a computerized item selection procedure. Behavior Research Methods and Instrumentation 6, 1 (Jan. 1974), 57-58.

119. Reynolds, C. H. Notes on estimating and other science fiction. Data Processing Magazine 9, 6 (June 1967), 44-45

120. Reynolds, C. H. More on estimating. Data Processing Magazine 9, 9 (Sept. 1967), 34.

121. Reynolds, C. H. The problem in perspective. Proc. of ACM National Conf., Brandon/Systems Press, Princeton, N.J., Aug. 1968, pp. 334-337.

122. Rhodes, J. J. Modular, planning and control. Computer Bulletin 14, 9 (Sept. 1970), 320-325.

123. Royce, W. W. Software requirements analysis: sizing and costing. in Practical Strategies for Developing Large Software Systems. E. Horowitz (Ed.), Addison-Wesley, Reading, Mass, 1975.

124. Rutter, J. H. Analysis of prediction process model. Operations Research Report A01-74 U.S. Air Force Data Systems Design Center, Gunter AFS Ala., Mar. 1974.

125. Sackman, H. Computers, System Science and Evolving Society. John Wiley and Sons, Inc., New York, 1967.

126. Sackman, H. Man-Computer Problem Solving. Auerbach, Inc., Princeton, N.J., 1970.

127. Schlaiffer, R. Probability and Statistics for Business Decisions. McGraw-Hill, New York, 1959.

128. Schröder, H. J. Making project management work. Management Review 59, 12 (Dec. 1970), 24-28.

129. Schwartz, J. I. Construction of software: problems and practicalities. in Practical Strategies for Developing Large Software Systems. E. Horowitz (Ed.), Addison-Wesley, Reading, Mass., 1975, pp. 15-53.

130. Scott, R., and Simmons, D. Programmer productivity and the Delphi technique. Datamation 20, 5 (May 1974), 71-73.

131. Shaw, J. C., and Atkins, W. Managing Computer System Projects. McGraw-Hill, New York, 1970.

132. Sherman, R. The people problem. Data Processing 14, 2 (Mar.-Apr. 1972), 104-107.

133. Sherman, R. Keeping up the pace. Data Processing 14, 5 (Sept.-Oct. 1972), 354-356.

134. Sherman, R. Evaluation and maintenance. Data Processing 14, 6 (Nov.-Dec. 1972), 424-426.

135. Slaughter, J. B. Understanding the software problem. Proc. AFIPS SJCC Vol. 40, AFIPS Press, Montvale, N.J., 1974, pp. 333-336.

136. Smith, D. An organization for successful project management. Proc. AFIPS SJCC Vol. 40, AFIPS Press, Montvale, N.J., 1972, pp. 129-140.

137. Smith, R. Structured programming series vol. XI: estimating software project resource requirements. Tech. Rep. RADC-TR-74-300, Rome Air Development Center, Griffiss AFB, N.Y., Jan. 1975.

138. Smith, R. Structured programming series vol. IX: management data collection and reporting. Tech. Rep. AD A008 640, Defense Documentation Center, Alexandria, Va., Oct. 1974.

139. Sobel, S. A computerized technique to express uncertainty in advanced system cost analysis. Tech. Rep. AD 624 894, Defense Documentation Center, Alexandria, Va., Nov. 1965.

140. SOFTECH Inc. Support software planning study. Tech. Rep. AD A012 964, Defense Documentation Center, Alexandria, Va., Mar. 1974.

141. Software costs: rising concern. Industrial Research 17, 7 (July 1975), 26.

142. Sullivan, J. E. Engineering of quality software systems: measuring the complexity of computer software. Tech. Rep. AD A007 770, Defense Documentation Center, Alexandria, Va., Jan. 1975.

143. Szweda, R. A. Information Processing Management. Auerbach Inc., Princeton, N.J., 1972.

144. Taylor, V. K. Problems in software development. Tech. Rep. AD 774 857, Defense Documentation Center, Alexandria, Va., Apr. 1973.

145. Teichroew, D., and Sayani, H. Automation of system building. Datamation 17, 8 (Aug. 1971), 25-30.

146. Thayer, R. RACD R & D program in computer language controls and software engineering techniques. Tech. Rep. AD 778 836, Defense Documentation Center, Alexandria, Va., Apr. 1974.

147. Tsichritzis, D. Project management. in Advanced Course on Software Engineering, F. L. Bauer (Ed.), Springer-Verlag, Berlin, 1973, pp. 374-383.

148. U.S. Dept. of the Air Force. Planning and Resource Management Information System. U.S. Air Force Data Systems Design Center Manual 300-405, Gunter AFS, Ala., Jan. 1973.

149. U.S. Dept. of the Army. Introduction to automatic data processing resource estimating procedures (ADPREP). Tech. Bul. TB 18-19-1, Headquarters Dept. of the Army, Washington, D.C., Apr. 1969.

150. Wadsworth, M. D. EDP Project Management Controls. Prentice-Hall, Englewood Cliffs, N.J., 1972.

151. Weinberg, G. M. The Psychology of Computer Programming. Van Norstrand Reinhold Co., New York, 1971.

152. Weinwurm, G. F., and Zagorski, H. J. Research into the management of computer programming: a transitional analysis of cost estimating techniques. Tech. Rep. AD 631 259, Defense Documentation Center, Alexandria, Va., Nov. 1965.

153. Weinwurm, G. F. Data elements for a cost reporting system for computer program development. Tech. Rep. AD 637 804, Defense Documentation Center, Alexandria, Va., Aug. 1966.

154. Weinwurm, G. F. Managing the economics of computer programming. Proc. of 23rd ACM National Conf., Brandon/System Press, Princeton, N.J., 1968, pp. 329-332.

155. Weiss, D. The MUDD report: a case study of Navy software development practices. Tech. Rep. AD A010 818, Defense Documentation Center, Alexandria, Va., May 1975.

156. Williams, R. D. Managing the development of reliable software. Proc. of 1975 International Conf. on Reliable Software, Apr. 1975, pp. 3-8.

157. Willmorth, N. E., and Mathis, N. S. Software milestone measurement. Tech. Rep. AD 775 305, Defense Documentation Center, Alexandria, Va., Nov. 1973.

158. Wolverton, R. W. The cost of developing large-scale software. IEEE Transactions on Computers 23, 6 (June 1974), 615-636.

159. Yearsley, R. B., and Graham, G. M. Handbook of Computer Management. Halsted Press, New York, 1973.

APPENDIX 1

STANDARD PHASE DEFINITIONS

Bxxx: FEASIBILITY STUDY: Involves the tasks associated with Economic Analysis, Data Automation Requirement (DAR) Preparation, Data Project Directive (DPD) Preparation, and Data Project Plan (DPP) Preparation as prescribed by AFM 300-12.

Cxxx: SYSTEM ANALYSIS: Involves those detailed tasks associated with System Analysis in the design or modification of a major automated data system (ADS).

Dxxx: PROGRAMMING: Involves those detailed tasks associated with the creation of a computer program.

Exxx: SYSTEM TEST: Involves the tasks associated with testing the entire ADS or a specific subsystem of an ADS.

F5xx: PRELIMINARY DOCUMENTATION--VOLUME I: Involves the writing and preparation for publication of the preliminary version of an AFM 171 series volume I manual.

F6XX: PRELIMINARY DOCUMENTATION--VOLUME II: Involves the writing and preparation for publication of the preliminary version of an AFM 171 series volume II manual.

F7xx: DOCUMENTATION-NON-B3500: Involves the writing and preparation for publication of the preliminary versions of AFM 171 series Volumes I, II, III, and IV manuals for other than B3500 equipment.

Gxxx: DOCUMENTATION--VOLUME III/FUNCTIONAL USER SUPPORT MANUAL: Involves the writing and preparation for publication of the preliminary version of an AFM 171 series volume III/Functional User Support Manual.

Hxxx: SYSTEM RELEASE/ENVIRONMENTAL SYSTEM TEST: This event group creates a reference point for all program/system releases and will facilitate the collection of resource expenditures by project for those functions performed by the Directorate of Systems Control (SC). This event group also provides the events necessary to identify mandatory milestone dates, such as entry into Environmental System Test and scheduled Air Force release dates. (See chapter 2, paragraph 2-11b for full explanation of procedures and use of this event group.)

Ixxx: IMPLEMENTATION/CONVERSION: Involves those tasks associated with the implementation or conversion of automated data systems and computer hardware and software.

X35.2: INDEPENDENT MINOR PROJECT: A project involving events which will require approximately 0 to 500 man-hours of effort for completion. (See chapter 2, paragraph 9c for full explanation of independent minor projects and their usage.)

APPENDIX 2

STANDARD ACTIVITY DEFINITIONS

B10.0: RESEARCH PROBLEM/REQUIREMENT: This event should be used to accumulate the time expended in all preliminary research and analysis associated with the problem/requirement under review. Included in this research is a general survey for the purpose of determining the operational and technical desirability of automating an application, as well as a complete description of the existing system. The survey should clarify the objectives and requirements of the proposed automated data system application/modification and explore all implications of the problem/requirement. Also included is the collection of data pertaining to existing procedures, policies, and the preparation of reports or position papers pertaining to these policies/procedures. Sources of information are current reports, files, work flow charts, organizational charts, and regulations/manuals.

B11.0: ACCOMPLISH ECONOMIC ANALYSIS: This event should be used to accumulate resource expenditures associated with the processes described in AFM 300-12 which relate to economic analyses for proposals which apply to an automated data system (ADS). This process is an iterative one in which there is no logical sequence of steps, but should generally address, to the degree practicable, the identification of the problem, description of the relevant environment, postulation of objectives, identification of assumptions and constraints, postulation of alternatives, cost estimates, identification of benefits, and comparison of alternatives.

B20.0: PREPARE DAR: Includes time expended in the actual preparation of the document which describes the need requiring management attention. The procedures in preparing this document, and the format, are contained in AFM 300-12.

B21.0: REVIEW/COORDINATE/APPROVE DAR: Includes time expended in the review and coordination process associated with each DAR prepared by AFDSDC. The estimated completion date of this event should indicate the date which it is anticipated that the DAR will be submitted for approval.

B30.0: PREPARE DPD: Includes time expended in the preparation of the DPD in accordance with AFM 300-12, attachment 2. The estimated start date for this event should be the anticipated approval date for the DAR, since the DPD must be issued within 30 days after approval of the DAR.

B31.0: REVIEW/COORDINATE/APPROVE DPD: Includes time expended by the Office of Primary Responsibility, each project participant, and each interested staff office in coordinating the proposed DPD. The estimated completion date for this event should be within 20 span days of the start date of DPD Preparation, and will indicate the anticipated

date which the data automation activity (SYD) approves the DPD and assigns a DPD number.

B40.0: PREPARE DPP: Includes time expended in the actual preparation of the DPP in the format prescribed by AFM 300-12, attachment 3.

B41.0: REVIEW/COORDINATE/APPROVE DPP: Includes time expended in review, coordination, and final approval of the DPP.

B50.0: SYSTEM DESIGN REVIEW: Includes time expended to prepare and conduct the initial Design Review Panel (DRP). During this review, the responsible Director will brief the DRP as to the purpose, objectives, and requirements of the system. The panel will review the system inputs, files, and records to be maintained; provide a concise portrayal of the developed system logic and processing steps (subsystem flow charts, edits, etc.); and describe processing requirements in narrative form. (NOTE: The findings and decisions made during this review establish the basis for future work; hence, this is considered to be the most crucial of the DRP meetings held during the development cycle. Each DRP meeting has a separate and unique objective. (See event C16.1.))

C01.0: DEFINE SYSTEM CONCEPTS: Includes time expended to review objectives established by management during the Feasibility Study, and to refine these objectives into the detailed definition of information requirements needed to complete system analysis.

C02.0: DEFINE INTERFACE/INTEGRATION REQUIREMENTS: Includes time expended to identify and document requirements for extraction of information from other systems, contribution of information to other systems, and extent of interaction with other systems.

C03.0: REVIEW/COORDINATE INTERFACE/INTEGRATION REQUIREMENTS: Includes time expended during that period, subsequent to the definition of interface/integration definition, when the definition is subjected to review by offices of primary and corollary interests. Includes any TDY, participation in meetings, and preparation and coordination of correspondence leading to final approval of the definition.

C04.0: IDENTIFY DATA BASE REQUIREMENTS/ELEMENTS/CODES: Includes time expended to identify data base elements required in the system activity.

C05.0: DEFINE DETAILED SYSTEM REQUIREMENTS AND OBJECTIVES: Includes time expended to identify and document detailed design and performance requirements for functional elements of information processing. Includes formatting of tables, tape layouts, labels, and other intermediate output media, and descriptions and proposed definitions for layouts and media requirements.

C06.0: FLOW CHART SYSTEM PROCESSES: Includes time expended to portray the flow of documents and information through the proposed system. Involves a review of flow charting accomplished during the Feasibility Study and the addition of details developed in actual system analysis.

C07.0: DEFINE INPUT DATA (MEDIA AND VOLUME): Includes time expended to format documents used to capture data for input to the computer.

Associated with this task is the requirement for structuring and coding data elements; also, structuring and formatting intermediate and master files, identifying input media, and estimating data volume to be anticipated.

C08.0: IDENTIFY TIMING FACTORS (SIMULATION) (SYO): Includes time expended to test system concepts; to identify timing factors, by use of a mathematical model; to assist in the selection of the most effective and efficient method of accomplishing a given task; to validate or disapprove hypotheses; or to use one computer system to mimic the operations of another.

C09.0: ANALYSE SYSTEM OPTIMIZATION: Includes time expended in a critical review of proposed inputs, outputs, objectives, and overall system requirements to affirm that the proposed system will most effectively and efficiently satisfy management goals.

C10.0: DEFINE AUDIT TRAIL REQUIREMENTS: Includes time expended to identify system elements requiring audit trail consideration, to provide features to allow association of records with original, to develop controls needed to assure complete transaction processing and record balancing, and to assure capability to recreate records damaged, destroyed, or lost during processing. (Involves timely Auditor General Representative's Office coordination.)

C11.0: COORDINATE SYSTEM WITH USER: Includes time expended in [illegible], participation in meetings, preparation and coordination of correspondence leading to user approval of the general system.

C12.0: DOCUMENT SYSTEM SPECIFICATIONS: Includes time expended to document general system information, specifications, and resource requirements that have been reviewed, refined, and coordinated during the user review phase of system analysis.

C13.0: DEFINE DETAIL SYSTEM PROCESSES: Includes time expended to identify, in proper format, the detailed requirements for system data editing, input and output formats, table and file formats, logical and arithmetic manipulation, data elements and codes, and control break data elements.

C14.0: PREPARE DETAIL SYSTEM FLOW CHARTS: Includes time expended to graphically depict elements described under Event C13.0. Involves preparation of a schematic diagram portraying the flow of data linking computer runs. This will be the working copy of the general system flow chart to be placed in volume I of either the Automated Management Supporting Data System (AMSDS) documentation or in general system documentation required by AFM 171-10.

C15.0: PREPARE AND REVIEW PROGRAM SPECIFICATIONS: Includes time expended to identify the detailed requirements for system data editing, input and output formats, table and file formats, logical and arithmetic manipulations, and data elements. Also includes time expended in obtaining a complete understanding between analyst(s) and programmer(s) involved in the system design effort.

C16.0: PREPARE SYSTEM TEST DATA: Includes time expended to devise meaningful test situations and conditions which will thoroughly exercise all programs and logic in the system. Anticipated test results must also be incorporated in this plan. The plan will also consider resource requirements for the test (personnel and equipment) and provide a schedule for the activities to be performed during the test; i.e., the tests themselves, time allotted for revisions, validation of documentation, etc., all time spent in accumulating or preparing test data to exercise the logic, editing, and output requirements of the entire system of interacting programs. Also, includes time expended in keypunching this data. Requirements established will be subject to revision during programmer desk check event.

C16.1: SYSTEM DESIGN REVIEW: Includes time expended to prepare and conduct the DRP which is required prior to formal programming. During this review, the responsible Director will present the final design plan to the panel, addressing the system processing requirements and explaining any differences between this panel's final design and the approval design that emerged from the previous review. The DRP will insure that major deviations are reconciled prior to final approval of the system.

D01.0: PROGRAM ANALYSIS/FLOW CHART: Includes time expended by the programmer to study the requirements of the program as outlined in the specifications and to formulate the detailed logic needed to satisfy these requirements.

D02.0: CODE: Includes time expended to translate detail block diagrams or required specifications into a specific programming language.

D03.0: KEYPUNCH: Includes time expended in assigning coded data to keypunch personnel, actual keypunching, and all actions leading to keypunch completion and verification.

D04.0: COMPILE/ASSEMBLE: Includes time expended in preparing the program for compilation or assembly.

D05.0: DESK CHECK/DEBUG: Includes time expended by the programmer, at his work place, checking his logic, coded instructions, and keypunching prior to initial program compilation. Also includes time expended to identify and correct logic, coding, and data errors detected during compilation.

D07.1: PREPARE TEST DATA: Includes time expended to devise meaningful test situations and conditions which will thoroughly exercise all program subroutines and program logic.

D09.0: TEST PROGRAM: Includes time expended to fully test program, all subroutines, and program logic. Also includes time expended to correct errors detected during the test.

D12.0: TEST PROGRAM IN SYBSYSTEM: Includes time expended to test the program in the environment where interaction with other programs of the system is required. Includes time expended to correct errors detected during the test.

D13.0: PREPARE PROGRAM FOLDER: Includes time expended in preparing the program folder (volume IV-B3500).

E01.0: TEST SYSTEM: Includes time expended to prepare, set up, and observe a system test and to collect test inputs (cards, tapes, and instructions) and outputs (dumps, reports, printout messages, etc.).

E02.0: DEBUG SYSTEM: Includes time expended, subsequent to initial system test, to analyze and evaluate test results; to generate recommendations for system or computer program documentation or specification changes; to prepare evaluation reports; and to correct errors, design deficiencies, or documentation inaccuracies revealed by the test.

E05.0: USER REVIEW/COORDINATION/APPROVAL: Includes time expended, subsequent to system debugging, in a review of the overall system with the user(s). Also includes an examination of originally established concepts and goals, leading to a mutual agreement between designer and user that established requirements are satisfied by the system.

E07.0: OPERATIONAL TEST/OPR APPROVAL: Includes time expended to obtain OPR approval for the system, including operational testing if required. Also includes the time spent TDY, participating in operational field tests, participating in meetings, and preparing/coordinating correspondence leading to OPR acceptance of the system.

E08.0: PREPARE SYSTEM FOR RELEASE TO SC: Includes time expended to prepare and coordinate a system change package for submission to the Directorate of Systems Control (SC). Necessary items include AF Forms 636 and 673, AFDSDC Forms 14 and 31, all test data, test run sheets, program Select Cards, etc., as required by AFDSDCR 171-9.

F50.0: ANNOUNCE DOCUMENTATION IN USAF PUBLICATIONS BULLETIN: Includes time required to formally announce supporting documentation prior to all new system releases. This event will also serve as an initial reminder when documentation efforts begin. This announcement must be made early enough that it appears in the USAF Publications Bulletin for a period of 45 to 65 days prior to distribution of new system documentation.

F52.0: PREPARE VOLUME I DOCUMENTATION: Includes time expended to draft general system instructions for formal publication. Includes system operation management documentation as defined by AFM 171-100, paragraph 6-4, or section A documentation as defined by AFM 171-10, paragraph 010203 (general), 050102 (B263), or 060102 (H800/200), as applicable.

F53.0: TYPE VOLUME I DRAFT: Includes time expended to format, review, correct, and type the working draft of volume I documentation.

F54.0: COORDINATE VOLUME I WITH AIR STAFF OPR: Includes time expended to review working draft or documentation, to meet with functional area representatives concerning the draft, to prepare correspondence, to travel, if required to accomplish the review, and to obtain initial Air Staff OPR approval.

F55.0: PREPARE VOLUME I FOR QUALITY CONTROL/ENVIRONMENTAL SYSTEM TEST: Includes time expended in typing final draft copy, final review, making corrections, and preparing necessary forms for submission to SCCQ.

F57.0: CORRECT VOLUME I: Includes time expended to make corrections to the formatted draft as a result of Quality Control/Environmental System Test, and to discuss proposed or required corrective action with personnel concerned.

F62.0: PREPARE VOLUME II DOCUMENTATION: Includes time expended to revise and define detailed system specifications and program documentation as required by AFM 171-100, chapter 6, paragraph 6-5.

F63.0: TYPE VOLUME II DRAFT: Includes time expended to prepare detailed specifications for formal publication. Includes information required by AFM 171-100, chapter 6, paragraph 6-5.

F64.0: COORDINATE VOLUME II WITH AIR STAFF OPR: Includes time expended to review the working draft of the documentation, to meet with functional area representatives concerning the draft, to prepare correspondence, and to travel, if required, to accomplish the review.

F65.0: PREPARE VOLUME II FOR QUALITY CONTROL/ENVIRONMENTAL SYSTEM TEST: Includes time expended in typing final draft copy, final review, making corrections, and preparation of necessary forms for submission to SCCQ.

F67.0: CORRECT VOLUME II: Includes span days and hours required to make any corrections to the formatted draft as a result of Quality Control/ Environmental System Test, and to discuss proposed or required corrective action with personnel concerned.

F72.0: PREPARE DOCUMENTATION: Includes time expended to develop and maintain appropriate system and program folders at the development center. These folders serve as a basis for the documentation required by AFM 171-10. Includes time expended to develop and maintain the manuals, to gather, organize, and proofread the material, and other related tasks.

F73.0: TYPE DRAFT: Includes time expended to hand-write, format, review, correct, and type a working draft. Also includes coordination.

F74.0: COORDINATE WITH AIR STAFF OPR: Includes time expended to complete a review of the documentation and to obtain OPR approval for system release. Includes making the corrections/revisions resulting from the OPR review.

F75.0: PREPARE FOR QUALITY CONTROL/ENVIRONMENTAL SYSTEM TEST: Includes time expended in typing final draft copy, final review, making corrections, and preparation of necessary forms for submission to SCCQ.

F77.0: CORRECT DOCUMENTATION: Include span days and hours required to make any corrections to the formatted draft as a result of Quality Control/Environmental System Test, and to discuss proposed or required corrective action with personnel concerned.

G12.0: PREPARE VOLUME II/FUNCTIONAL USERS' SUPPORT MANUAL DOCUMENTATION: Includes time expended to revise and define detailed system

specifications and program documentation as required by AFM 171-100, paragraph 6-6.

G13.0: TYPE VOLUME III/FUNCTIONAL USERS' SUPPORT MANUAL DRAFT: Includes time expended to prepare information required by AFM 171-100, paragraph 6-6.

G14.0: COORDINATE VOLUME III/FUNCTIONAL USERS' SUPPORT MANUAL WITH AIR STAFF OPR: Includes time expended to send or hand-carry the draft of the documentation to assure acceptability of the draft for formatting an in-depth content review, to identify elements which are not in consonance with established policies, procedures, or standards, and to properly format the material.

G15.0: PREPARE VOLUME III/FUNCTIONAL USERS' SUPPORT MANUAL FOR QUALITY CONTROL/ENVIRONMENTAL SYSTEM TEST: Includes time expended in typing final draft copy, final review, making corrections, and preparation of necessary forms for submission to SCCQ.

G17.0: CORRECT VOLUME III/FUNCTIONAL USERS' SUPPORT MANUAL: Includes span days and hours required to make any corrections to the formatted draft as a result of the Environmental System Test, and to discuss proposed or required corrective action with personnel concerned.

H01.0: PRODUCT REVIEW: This event will include time expended in the final product review, required by AFDSDCR 171-11, prior to releasing system documentation and programs for system release/Environmental System Test.

H02.0: ENVIRONMENTAL SYSTEM TEST, PHASE I: This is a milestone event and will not be used to accumulate resource expenditures. It is established solely to indicate the date the Project Manager expects to submit the system/program package to SCCQ for Environmental System Test, Phase I. Phase I will normally be performed on all B3500 system change packages prior to their being released to the field. Non-B3500 systems use Event H04.0, Environmental System Test, Phase II. Span days for this event will be 25, to correspond with the established system release cycle. Subevents H02.1 through H02.5 will be used by SCCQ to record resource expenditures in conducting Environmental System Test, Phase I and they will not appear on the Active Schedule.

H02.1: SCCR FUNCTIONS: This event will include time expended by SCCR in systems packages receipt, initiating the administrative controls, review for completeness, and all the subsequent administrative efforts involved in preparing the system package for release.

H02.2: PROGRAM/SYSTEM REVIEW: This event will include the time associated with the analysis of programmed input/output, file condition, processing inter-relationship, and environmental conditions to isolate problem areas and aid in resolving difficulties. Also included will be the time expended in developing the test procedures and insuring that the programs are compatible with documentation.

H02.3: PROGRAM/SYSTEM TEST: This event will be utilized to record the time expended in testing, in an integrated systems environment, new and revised computer programs to insure proper interface of program elements, compatibility with operator procedures, and conformance to USAF standards.

H02.4: DEBRIEFING TEST RESULTS: This event will be used by Quality Control personnel to record the time associated with advising the development agency of the results of the specific program/system evaluation.

H02.5: DOCUMENTATION REPRODUCTION: This event has been established to record the man-hours required in preparing the preliminary documentation to accompany the program/systems release.

H04.0: ENVIRONMENTAL SYSTEM TEST, PHASE II: This event will be used to record all the time expended in the evaluation of computer programs, documentation, and output products (including their responsiveness to management's needs) in a live environment at one or more selected Air Force bases prior to formal world-wide release.

H99.9: AIR FORCE RELEASE: This event will be used by the Project Manager to project his best estimate of when he expects the program/system to be released Air Force-wide. This is a milestone event and will not be used for accumulating productive hours. Every effort must be made to keep this date as accurate as possible, as it will be used for planning purposes throughout AFDSDC. It may also be used in periodic releases to the data processing installations to keep them advised of planned systems releases.

K99.8: FORMAL DOCUMENTATION: This event will include all time expended by the Directorate of Systems Control in the formal publication of system documentation, when that effort is associated with small PARMIS projects established using Event Groups X35.0, X35.1, and X35.2. This event is required in all projects established in these event groups. A full explanation of the procedures associated with the use of this event is contained in chapter 2, paragraph 2-11 of this manual.

K99.9: RELEASE: This event will include all time expended by the Directorate of Systems Control in conducting Environmental System Test, when that effort is associated with small PARMIS projects established using Event Groups X35.0, X35.1, and X35.2. This event is required in all projects established in these event groups. A full explanation of the procedures associated with the use of this event is contained in chapter 2, paragraph 2-11 of this manual.

I01.0: WRITE IMPLEMENTATION/CONVERSION PLAN: Includes time expended to plan and schedule the total Implementation/Conversion project, to coordinate the plan with Hq USAF, and to update the plan as experience is gained during actual Implementation/Conversion.

I02.0: WRITE TRAINING PLAN: Includes time expended to detail the training requirements of all personnel involved in the Implementation/Conversion effort.

I03.0: TRAIN IMPLEMENTATION/CONVERSION TEAM: Includes time expended to train Implementation/Conversion Team representatives from both the AFDSDC and the major commands.

I04.0: TRAIN BASE PERSONNEL: Includes time expended to train both the OPR and data automation personnel at each lead base on actions required during the Implementation/Conversion effort. This event could appear numerous times to reflect the schedule of lead base training requirements.

I05.0: SITE PREPARATION: Includes time expended by Design Center personnel to inspect or assist in the base site preparation when new equipment installations or large equipment enhancements are required for system implementation. This event should include time spent on TDY for site inspection or preparation assistance.

I06.0: ACCEPTANCE TESTING: Includes time expended to run diagnostic programs and routines to insure that the installed hardware fulfills all expected requirements.

I07.0: PRECONVERSION/DATA COLLECTION: Includes time expended to purify data bases, perform dummy edits, and accomplish all necessary data collection at either the base site or at AFDSDC.

I08.0: CONVERT FILES AT BASE: Includes time expended to assist in the conversion of files at each lead base. This event could appear numerous times to reflect the schedule of the file conversion of each individual lead base.

I09.0: POSTCONVERSION: Includes time expended in follow-up visitations to the base site to insure that all new procedures are working as planned.

I10.0: LOAD/IMPLEMENT SYSTEM AT BASE: Includes time expended to upload the new system/modification at the lead base. This event may appear numerous times to include time expended at each lead base to get the system/subsystem operating correctly.

I11.0: POSTINSTALLATION SYSTEM SURVEY: Includes time expended for any needed final visit to survey operations sometime after the new system/subsystem has been in full operation.

APPENDIX 3

SAMPLE SEQUIN OUTPUT

```
SEQUENTIAL ITEM NOMINATOR                                SAMPLE SIZE       39.
USER ***PHILIP GENRING                                   DATE USED 02/07/76
TITLE ***EVENT ESTIMATING RELATIONSHIP TO PREDICTING PROJECT ESTIMATES
COMMENTS ***    1.00*SDD 1/2
COMMENTS ***

THE FOLLOWING ITEMS HAVE DIFFICULTIES AS INDICATED AND
WILL NOT BE CONSIDERED IN ANY SUBSEQUENT ANALYSIS.
 ITEM          DIFFICULTY
    3              0.0
   56              0.0
   57              0.0
   58              0.0
   65              0.0
   67              0.0
   68              0.0
   69              0.0
   70              0.0
   71              0.0
   72              0.0
```

```
NUMBER OF CRITERION
   1          2
MEANS
   1          2
14.077   866.359

STANDARD DEVIATIONS
   1          2
 9.740   *******

INTERCRITERION CORRELATION
   1          2
1   1.000    -0.186
2                1.000
```

SEQUENTIAL ITEM NOMINATOR SAMPLE SIZE 39.
USER ***PHILIP GEHRING DATE USED 02/07/76
TITLE ***EVENT ESTIMATING RELATIONSHIP TO PREDICTING PROJECT ESTIMATES
COMMENTS *** 1.00*SDD 1/2
COMMENTS ***

SEQUENCE NUMBER	CRITERION NUMBER	ITEM NUMBER	ITEM DIFF	POINT BISERIAL	BISERIAL CORR	CUM CORR	CORR CHANGE	INTERNAL CONSISTENCY	CONSISTENCY CHANGE	TEST MEAN	TEST SD	SD ERROR MEASUREMENT	JOINT REVAL	REVAL CHANGE
1.	INT 1.	26.	0.205	0.894	1.254	0.894	0.894	0.0	0.0	0.205	0.409	0.0	0.0	0.0
2.	INT 1.	62.	0.179	0.740	1.070	0.919	0.024	0.741	0.741	0.385	0.711	0.362	0.681	0.681
3.	INT 1.	48.	0.077	0.488	0.890	0.955	0.036	0.632	-0.110	0.462	0.822	0.499	0.603	-.078
4.	INT 1.	16.	0.256	0.869	1.162	0.964	0.009	0.801	0.169	0.718	1.213	0.541	0.773	0.169
5.	INT 1.	17.	0.026	0.336	0.879	0.962	-0.002	0.776	-0.025	0.744	1.272	0.601	0.747	-.026
6.	INT 1.	61.	0.410	0.693	0.865	0.962	-0.000	0.812	0.035	1.154	1.631	0.708	0.781	0.034
7.	INT 1.	15.	0.205	0.689	0.967	0.968	0.006	0.839	0.027	1.359	1.912	0.768	0.812	0.031
8.	INT 1.	34.	0.846	0.358	0.538	0.977	0.009	0.820	-0.018	2.205	2.028	0.860	0.801	-.010
9.	INT 1.	4.	0.026	-.086	-.224	0.981	0.004	0.793	-0.027	2.231	2.006	0.913	0.778	-.023
10.	INT 1.	18.	0.103	0.701	1.174	0.980	-0.001	0.824	0.031	2.333	2.228	0.934	0.808	0.030
11.	INT 1.	13.	0.308	0.775	1.004	0.979	-0.001	0.856	0.032	2.641	2.600	0.987	0.838	0.030
12.	INT 1.	32.	0.436	0.574	0.713	0.981	0.003	0.861	0.006	3.077	2.887	1.075	0.846	0.008
13.	INT 1.	42.	0.308	0.018	0.023	0.984	0.002	0.827	-0.034	3.385	2.889	1.200	0.814	-.032
14.	INT 1.	63.	0.128	0.643	1.013	0.986	0.003	0.843	0.016	3.513	3.102	1.227	0.832	0.018
15.	INT 1.	35.	0.103	0.349	0.585	0.987	0.001	0.844	0.001	3.615	3.209	1.265	0.834	0.002
16.	INT 1.	31.	0.410	0.666	0.831	0.988	0.001	0.860	0.016	4.026	3.543	1.324	0.850	0.016
17.	INT 1.	37.	0.205	-.235	-.330	0.990	0.002	0.829	-0.031	4.231	3.437	1.421	0.821	-.029
18.	INT 1.	44.	0.026	-.103	-.268	0.991	0.001	0.821	-0.008	4.256	3.416	1.446	0.814	-.007
19.	INT 1.	52.	0.051	0.458	0.947	0.992	0.000	0.828	0.007	4.308	3.518	1.460	0.821	0.007
20.	INT 1.	10.	0.256	0.649	0.868	0.993	0.001	0.844	0.016	4.564	3.803	1.502	0.838	0.017
21.	INT 1.	55.	0.026	0.134	0.350	0.993	0.000	0.842	-0.002	4.590	3.823	1.519	0.836	-.001
22.	INT 1.	5.	0.103	0.454	0.761	0.994	0.001	0.848	0.006	4.692	3.961	1.544	0.843	0.006
23.	INT 1.	2.	0.026	-.086	-.224	0.994	0.000	0.843	-0.005	4.718	3.947	1.563	0.838	-.005
24.	INT 1.	36.	0.333	0.441	0.565	0.993	-0.001	0.848	0.005	5.051	4.161	1.622	0.843	0.004
25.	INT 1.	47.	0.103	0.639	1.071	0.994	0.001	0.859	0.010	5.154	4.356	1.638	0.853	0.011
26.	INT 1.	53.	0.051	0.349	0.722	0.995	0.001	0.861	0.002	5.205	4.432	1.654	0.856	0.003
27.	INT 1.	11.	0.179	0.816	1.180	0.995	0.000	0.876	0.015	5.385	4.750	1.675	0.871	0.015
28.	INT 1.	45.	0.308	0.671	0.869	0.995	0.000	0.885	0.009	5.692	5.064	1.716	0.881	0.010
29.	INT 1.	14.	0.564	0.507	0.630	0.996	0.001	0.889	0.004	6.256	5.315	1.773	0.885	0.004
30.	INT 1.	1.	0.026	-.086	-.224	0.996	-0.000	0.886	-0.003	6.282	5.301	1.790	0.882	-.003
31.	INT 1.	29.	0.795	0.348	0.487	0.997	0.001	0.887	0.001	7.077	5.441	1.832	0.884	0.001
32.	INT 1.	54.	0.051	0.167	0.347	0.997	-0.000	0.886	-0.001	7.128	5.478	1.849	0.883	-.001
33.	INT 1.	59.	0.026	-.187	-.489	0.996	-0.001	0.883	-0.003	7.154	5.451	1.866	0.879	-.004
34.	INT 1.	23.	0.256	0.631	0.844	0.996	-0.001	0.890	0.007	7.410	5.734	1.900	0.886	0.007
35.	INT 1.	41.	0.231	0.129	0.176	0.996	0.000	0.886	-0.004	7.641	5.788	1.951	0.883	-.004

SEQUENTIAL ITEM NOMINATOR — SAMPLE SIZE 39.
USER ***PHILIP GEHRING — DATE USED 02/07/76
TITLE ***EVENT ESTIMATING RELATIONSHIP TO PREDICTING PROJECT ESTIMATES
COMMENTS *** 1.00*SDD 1/2
COMMENTS ***

SEQUENCE NUMBER	CRITERION NUMBER		ITEM NUMBER	ITEM DIFF	POINT BISERIAL	BISERIAL CORR	CUM CORR	CORR CHANGE	INTERNAL CONSISTENCY	CONSISTENCY CHANGE	TEST MEAN	TEST SD	SD ERROR MEASUREMENT	JOINT REVAL	REVAL CHANGE
36.	INT	1.	49.	0.051	0.131	0.271	0.996	-0.000	0.886	-0.001	7.692	5.818	1.968	0.882	-.001
37.	INT	1.	40.	0.231	0.489	0.669	0.996	-0.000	0.890	0.004	7.923	6.028	2.003	0.886	0.004
38.	INT	1.	12.	0.205	0.749	1.050	0.996	0.000	0.898	0.008	8.128	6.334	2.026	0.894	0.008
39.	INT	1.	6.	0.077	0.298	0.543	0.996	0.000	0.898	0.001	8.205	6.412	2.044	0.895	0.001
40.	INT	1.	64.	0.077	0.458	0.835	0.996	-0.000	0.901	0.003	8.282	6.537	2.058	0.897	0.002
41.	INT	1.	43.	0.231	0.065	0.089	0.996	0.000	0.897	-0.004	8.513	6.565	2.107	0.894	-.004
42.	INT	1.	28.	0.538	0.585	0.725	0.997	0.000	0.902	0.005	9.051	6.859	2.151	0.899	0.005
43.	INT	1.	25.	0.359	0.711	0.901	0.996	-0.000	0.908	0.006	9.410	7.206	2.186	0.905	0.006
44.	INT	1.	9.	0.128	0.396	0.623	0.997	0.000	0.910	0.002	9.538	7.341	2.208	0.906	0.002
45.	INT	1.	21.	0.359	0.489	0.619	0.996	-0.000	0.912	0.002	9.897	7.580	2.249	0.909	0.002
46.	INT	1.	60.	0.205	-.130	-.182	0.997	0.000	0.907	-0.005	10.103	7.525	2.297	0.904	-.005
47.	INT	1.	27.	0.179	-.205	-.297	0.997	0.000	0.901	-0.006	10.282	7.444	2.341	0.898	-.006
48.	INT	1.	46.	0.077	0.458	0.835	0.997	-0.000	0.903	0.002	10.359	7.569	2.353	0.900	0.002
49.	INT	1.	30.	0.333	0.554	0.709	0.997	0.000	0.907	0.004	10.692	7.834	2.389	0.904	0.004
50.	INT	1.	39.	0.282	0.528	0.695	0.997	0.000	0.910	0.003	10.974	8.074	2.422	0.907	0.003
51.	INT	1.	24.	0.462	0.164	0.203	0.997	0.001	0.908	-0.002	11.436	8.152	2.475	0.906	-.002
52.	INT	1.	8.	0.051	0.143	0.297	0.998	0.000	0.908	-0.000	11.487	8.182	2.488	0.906	0.000
53.	INT	1.	38.	0.667	0.226	0.290	0.998	0.000	0.907	-0.001	12.154	8.289	2.532	0.905	-.001
54.	INT	1.	19.	0.103	0.560	0.938	0.998	-0.000	0.910	0.003	12.256	8.463	2.544	0.908	0.003
55.	INT	1.	7.	0.077	0.318	0.580	0.998	0.000	0.910	0.001	12.333	8.548	2.558	0.909	0.001
56.	INT	1.	22.	0.308	0.480	0.622	0.998	-0.000	0.913	0.002	12.641	8.773	2.591	0.911	0.002
57.	INT	1.	51.	0.103	0.534	0.894	0.998	0.000	0.915	0.002	12.744	8.935	2.604	0.913	0.003
58.	INT	1.	33.	0.282	0.274	0.360	0.998	-0.000	0.915	-0.000	13.026	9.060	2.642	0.913	-.000
59.	INT	1.	50.	0.436	0.396	0.493	0.999	0.001	0.916	0.001	13.462	9.253	2.682	0.915	0.002
60.	INT	1.	66.	0.333	0.504	0.644	0.999	0.000	0.918	0.002	13.795	9.490	2.716	0.917	0.003
61.	INT	1.	20.	0.282	0.564	0.741	1.000	0.001	0.921	0.003	14.077	9.740	2.744	0.921	0.003

SEQUENTIAL ITEM NOMINATOR SAMPLE SIZE 39.
USER ***PHILIP GEHRING DATE USED 02/07/76
TITLE ***EVENT ESTIMATING RELATIONSHIP TO PREDICTING PROJECT ESTIMATES
COMMENTS *** 1.00*SDD 1/2
COMMENTS ***

SEQUENCE NUMBER	CRITERION NUMBER	ITEM NUMBER	ITEM DIFF	POINT BISERIAL	BISERIAL CORR	CUM CORR	CORR CHANGE	INTERNAL CONSISTENCY	CONSISTENCY CHANGE	TEST MEAN	TEST SD	SD ERROR MEASUREMENT	JOINT REVAL	REVAL CHANGE
1.	EXT 1.	37.	0.205	0.479	0.672	0.479	0.479	0.0	0.0	0.205	0.409	0.0	0.0	0.0
2.	EXT 1.	27.	0.179	0.390	0.564	0.549	0.070	0.411	0.411	0.385	0.633	0.486	0.226	0.226
3.	EXT 1.	66.	0.333	0.260	0.333	0.622	0.073	0.077	-0.333	0.718	0.759	0.729	0.048	-.178
4.	EXT 1.	60.	0.205	0.328	0.459	0.653	0.031	0.230	0.152	0.923	0.929	0.815	0.150	0.102
5.	EXT 1.	4.	0.026	-.034	-.090	0.655	0.003	0.149	-0.081	0.949	0.916	0.845	0.097	-.052
6.	EXT 1.	41.	0.231	0.055	0.076	0.662	0.007	-.046	-0.194	1.179	0.942	0.964	-.030	-.128
7.	EXT 1.	59.	0.026	-.081	-.211	0.662	0.000	-.131	-0.086	1.205	0.923	0.981	-.087	-.057
8.	EXT 1.	2.	0.026	-.034	-.090	0.651	-0.012	-.141	-0.009	1.231	0.931	0.994	-.092	-.005
9.	EXT 1.	29.	0.795	0.136	0.190	0.637	-0.014	-.065	0.076	2.026	1.038	1.071	-.041	0.050
10.	EXT 1.	44.	0.026	-.064	-.167	0.635	-0.001	-.123	-0.058	2.051	1.025	1.086	-.078	-.037
11.	EXT 1.	43.	0.231	0.329	0.450	0.630	-0.005	0.159	0.282	2.282	1.255	1.151	0.100	0.178
12.	EXT 1.	1.	0.026	-.034	-.090	0.624	-0.006	0.147	-0.013	2.308	1.260	1.164	0.091	-.009
13.	EXT 1.	55.	0.026	-.084	-.219	0.611	-0.012	0.134	-0.013	2.333	1.264	1.176	0.082	-.010
14.	EXT 1.	42.	0.308	0.022	0.029	0.599	-0.013	0.058	-0.076	2.641	1.308	1.269	0.035	-.047
15.	EXT 1.	49.	0.051	-.117	-.243	0.591	-0.008	-.019	-0.076	2.692	1.280	1.292	-.011	-.046
16.	EXT 1.	38.	0.667	0.256	0.328	0.581	-0.010	0.183	0.202	3.359	1.513	1.367	0.106	0.117
17.	EXT 1.	17.	0.026	-.062	-.161	0.567	-0.014	0.193	0.010	3.385	1.532	1.377	0.109	0.003
18.	EXT 1.	61.	0.410	0.181	0.227	0.548	-0.019	0.308	0.115	3.795	1.750	1.455	0.169	0.060
19.	EXT 1.	53.	0.051	-.079	-.164	0.528	-0.021	0.320	0.012	3.846	1.785	1.472	0.169	-.000
20.	EXT 1.	54.	0.051	-.090	-.187	0.507	-0.021	0.330	0.010	3.897	1.818	1.488	0.167	-.002
21.	EXT 1.	8.	0.051	-.091	-.188	0.488	-0.019	0.337	0.008	3.949	1.849	1.505	0.165	-.003
22.	EXT 1.	5.	0.103	-.089	-.149	0.462	-0.025	0.340	0.003	4.051	1.891	1.536	0.157	-.007
23.	EXT 1.	40.	0.231	0.070	0.096	0.435	-0.027	0.416	0.076	4.282	2.077	1.587	0.181	0.024
24.	EXT 1.	64.	0.077	-.124	-.226	0.411	-0.025	0.424	0.008	4.359	2.121	1.609	0.174	-.007
25.	EXT 1.	52.	0.051	-.100	-.207	0.386	-0.024	0.455	0.030	4.410	2.197	1.622	0.176	0.001
26.	EXT 1.	45.	0.308	0.108	0.140	0.358	-0.028	0.556	0.101	4.718	2.513	1.674	0.199	0.023
27.	EXT 1.	46.	0.077	-.120	-.218	0.334	-0.024	0.575	0.019	4.795	2.597	1.693	0.192	-.007
28.	EXT 1.	7.	0.077	-.112	-.204	0.310	-0.024	0.597	0.022	4.872	2.697	1.711	0.185	-.007
29.	EXT 1.	6.	0.077	-.121	-.221	0.286	-0.024	0.621	0.024	4.949	2.809	1.728	0.178	-.007
30.	EXT 1.	48.	0.077	-.102	-.186	0.265	-0.022	0.646	0.025	5.026	2.933	1.745	0.171	-.007
31.	EXT 1.	35.	0.103	-.153	-.257	0.242	-0.022	0.654	0.008	5.128	3.010	1.770	0.159	-.013
32.	EXT 1.	9.	0.128	-.082	-.129	0.220	-0.022	0.682	0.028	5.256	3.185	1.795	0.150	-.008
33.	EXT 1.	30.	0.333	-.070	-.085	0.201	-0.020	0.690	0.008	5.590	3.330	1.853	0.139	-.012
34.	EXT 1.	10.	0.256	-.052	-.069	0.182	-0.019	0.715	0.024	5.846	3.551	1.897	0.130	-.009
35.	EXT 1.	19.	0.103	-.139	-.234	0.164	-0.018	0.729	0.015	5.949	3.685	1.917	0.119	-.011

SEQUENTIAL ITEM NOMINATOR — SAMPLE SIZE 39.
USER ***PHILIP GEHRING — DATE USED 02/07/76
TITLE ***EVENT ESTIMATING RELATIONSHIP TO PREDICTING PROJECT ESTIMATES
COMMENTS *** 1.00*SDD 1/2
COMMENTS ***

SEQUENCE NUMBER	CRITERION NUMBER	ITEM NUMBER	ITEM DIFF	POINT BISERIAL	BISERIAL CORR	CUM CORR	CORR CHANGE	INTERNAL CONSISTENCY	CONSISTENCY CHANGE	TEST MEAN	TEST SD	SD ERROR MEASUREMENT	JOINT REVAL	REVAL CHANGE
36.	EXT 1.	18.	0.103	-.122	-.204	0.146	-0.018	0.752	0.022	6.051	3.879	1.933	0.110	-.010
37.	EXT 1.	51.	0.103	-.163	-.273	0.128	-0.017	0.763	0.011	6.154	4.010	1.953	0.098	-.012
38.	EXT 1.	32.	0.436	-.065	-.081	0.113	-0.016	0.781	0.018	6.590	4.284	2.005	0.088	-.010
39.	EXT 1.	50.	0.436	-.079	-.098	0.097	-0.015	0.795	0.014	7.026	4.545	2.057	0.077	-.010
40.	EXT 1.	47.	0.103	-.152	-.254	0.084	-0.014	0.809	0.014	7.128	4.742	2.071	0.068	-.010
41.	EXT 1.	15.	0.205	-.120	-.168	0.070	-0.014	0.822	0.012	7.333	4.975	2.101	0.057	-.010
42.	EXT 1.	14.	0.564	-.114	-.142	0.056	-0.014	0.828	0.006	7.897	5.190	2.153	0.046	-.011
43.	EXT 1.	39.	0.282	-.125	-.164	0.043	-0.013	0.836	0.008	8.179	5.419	2.191	0.036	-.010
44.	EXT 1.	63.	0.128	-.177	-.279	0.031	-0.012	0.847	0.011	8.308	5.648	2.207	0.026	-.010
45.	EXT 1.	34.	0.846	-.230	-.346	0.015	-0.015	0.849	0.001	9.154	5.747	2.236	0.013	-.013
46.	EXT 1.	28.	0.538	-.403	-.499	0.019	0.004	0.853	0.004	9.692	5.952	2.285	0.016	0.003
47.	EXT 1.	31.	0.410	-.383	-.478	0.049	0.030	0.861	0.008	10.103	6.240	2.326	0.042	0.026
48.	EXT 1.	24.	0.462	-.312	-.387	0.074	0.025	0.855	-0.006	10.564	6.261	2.383	0.063	0.021
49.	EXT 1.	33.	0.282	-.302	-.397	0.095	0.021	0.854	-0.002	10.846	6.339	2.426	0.081	0.018
50.	EXT 1.	22.	0.308	-.242	-.313	0.110	0.015	0.857	0.003	11.154	6.515	2.465	0.094	0.013
51.	EXT 1.	20.	0.282	-.255	-.336	0.123	0.013	0.863	0.007	11.436	6.758	2.498	0.106	0.012
52.	EXT 1.	13.	0.308	-.260	-.336	0.134	0.011	0.873	0.010	11.744	7.100	2.527	0.117	0.011
53.	EXT 1.	25.	0.359	-.240	-.304	0.144	0.010	0.881	0.008	12.103	7.423	2.560	0.127	0.010
54.	EXT 1.	21.	0.359	-.201	-.255	0.153	0.009	0.884	0.003	12.462	7.636	2.598	0.135	0.008
55.	EXT 1.	36.	0.333	-.197	-.252	0.161	0.008	0.887	0.003	12.795	7.838	2.635	0.143	0.008
56.	EXT 1.	16.	0.256	-.264	-.354	0.168	0.007	0.895	0.008	13.051	8.211	2.655	0.150	0.008
57.	EXT 1.	23.	0.256	-.197	-.263	0.173	0.005	0.900	0.005	13.308	8.483	2.681	0.155	0.005
58.	EXT 1.	62.	0.179	-.215	-.311	0.176	0.004	0.905	0.005	13.487	8.772	2.697	0.160	0.004
59.	EXT 1.	12.	0.205	-.206	-.289	0.180	0.004	0.910	0.005	13.692	9.065	2.716	0.164	0.004
60.	EXT 1.	26.	0.205	-.233	-.327	0.183	0.003	0.916	0.006	13.897	9.425	2.730	0.168	0.004
61.	EXT 1.	11.	0.179	-.213	-.309	0.186	0.003	0.921	0.005	14.077	9.740	2.744	0.171	0.003

VITA

Lieutenant Colonel Philip F. Gehring, Jr., U.S. Air Force, was born on 26 February 1932, in Trenton, New Jersey, the son of Mr. and Mrs. Philip F. Gehring. He attended the U.S. Naval Academy and received a B.S. degree in Engineering in 1955. Lt. Col. Gehring has served 21 years in the U.S. Air Force with experience as a Logistics Officer, as a Crew Commander in the Strategic Air Command for the ATLAS E intercontinental ballistic missile, and as a Data Automation Staff Officer. His M.S. was taken in Information Science at the Georgia Institute of Technology in 1966. He is also an outstanding graduate of the Air War College. Lt. Col. Gehring is a past secretary of the Alpha Chapter of Upsilon Pi Epsilon and is a member of the Computer Society of IEEE and the Data Processing Management Association. His permanent address is: 7 Clarence Avenue, West End, New Jersey 07740.

The typist for this dissertation was Mrs. Joan Barry. Tables and graphs were drawn by Ms. Barbara Webb.