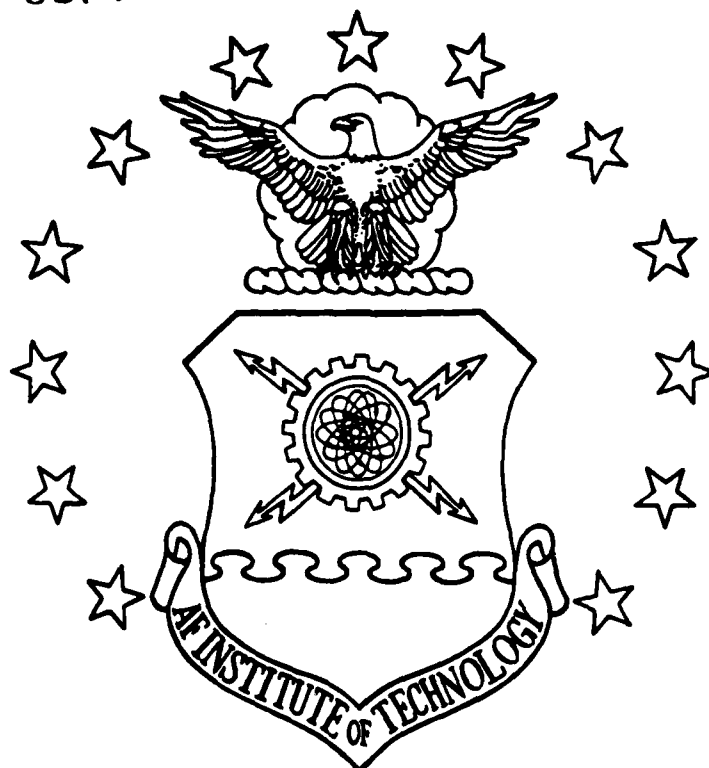


2

DTIC FILE COPY

AD-A201 628



AN ANALYSIS OF SCHEDULE DETERMINATION
IN SOFTWARE PROGRAM DEVELOPMENT
AND SOFTWARE DEVELOPMENT
ESTIMATION MODELS

THESIS

Crystal D. Blalock, B.S.
Captain, USAF

AFIT/GCA/LSY/88S-2

DTIC
SELECTED
DEC 20 1988
α
E

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

This document has been approved
for public release and sale the
distribution is unlimited.

88 12 20 48 037

2

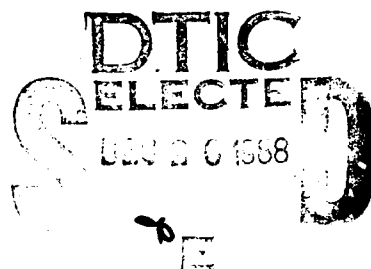
AFIT/GCA/LSY/88S-2

AN ANALYSIS OF SCHEDULE DETERMINATION
IN SOFTWARE PROGRAM DEVELOPMENT
AND SOFTWARE DEVELOPMENT
ESTIMATION MODELS

THESIS

Crystal D. Blalock, B.S.
Captain, USAF

AFIT/GCA/LSY/88S-2



Approved for public release; distribution unlimited

The contents of the document are technically accurate, and no sensitive items, detrimental ideas, or deleterious information is contained therein. Furthermore, the views expressed in the document are those of the author and do not necessarily reflect the views of the School of Systems and Logistics, the Air University, the United States Air Force, or the Department of Defense.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



AFIT/GCA/LSY/88S-2

**AN ANALYSIS OF SCHEDULE DETERMINATION
IN SOFTWARE PROGRAM DEVELOPMENT AND
SOFTWARE DEVELOPMENT ESTIMATION MODELS**

THESIS

**Presented to the Faculty of the School of Systems and Logistics
of the Air Force Institute of Technology**

Air University

**In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Cost Analysis**

Crystal D. Blalock, B.S.

Captain, USAF

September 1988

Approved for public release; distribution unlimited

Acknowledgements

Analyzing schedule in software development programs was a challenging endeavor. It required the inputs and assistance of key individuals throughout the field to make this research effort a success. I would like to acknowledge those individuals.

In order to determine the variables that contribute most to schedule in software development programs, interviews of program managers/engineers and other experts in the field had to be conducted. But, before those individuals could be interviewed, they had to be identified. I would like to thank Captain Joe Dean, Mr. Brad Donald, Mr. Dave Hansen, and Mr. Tom Bernard for supplying these names to me. I am also sincerely grateful to all the people I interviewed. I appreciate the time they took from their busy schedules to review my questionnaire and then talk to me personally.

To successfully analyze the software estimation models I chose for this thesis, I first had to have access to them. I am thankful to Professor Daniel V. Ferens for allowing me to use his copies of the models he has access to for academic purposes. I am also grateful to Aeronautical Systems Division for allowing me access to the PRICE-S™ model and to Mr. Jim Otte for providing me assistance in operating this model. Second, I needed data to input into these models in order to make a comparison. I am once again thankful to Captain Joe Dean for providing me with this data.

Finally, I would like to sincerely thank my friend, Captain Mark Pohlmeier for all the assistance he has given me while I worked on this research effort. He has been a source of knowledge, motivation, inspiration, comfort and friendship that I shall never forget.

Trademark Acknowledgements

PRICE-S is a trademark of RCA, Incorporated, a subsidiary of General Electric, Incorporated.

SLIM is a trademark of Quantitative Software Management, Incorporated.

SoftCost-R is a trademark of Reifer Consultants, Incorporated.

SPQR/20 is a trademark of Software Productivity Research, Incorporated.

System-3 is a trademark of Computer Economics Incorporated.

Table of Contents

	Page
Acknowledgements	ii
Trademark Acknowledgements	iv
List of Figures	viii
List of Tables	ix
List of Equations	x
Abstract	xi
I. INTRODUCTION	1
General Issue	1
Definitions	2
Computer Software Component (CSC)	2
Computer Software Configuration Item (CSCI)	3
Cost Model	3
Milestone	3
Schedule	3
Schedule Risk	3
Software Development Project/Program	4
Virtual Machine	4
Specific Problem	4
Justification	5
Assumptions	5
Research Objectives	6
Investigative Questions	7
Scope and Limitations	7
II. Literature Review	9
Software Development	9
Determinants of Schedule Risk	11
DOD Standard 2167	13
Software Development Estimation Models	16
The Constructive Cost Model (COCOMO)	17
PRICE-S	20
SLIM	24
SoftCost-R	26
SPQR/20	28
System-3	31
III. METHODOLOGY	34
Introduction	34
Question One	34
Question Two	34

	Page
Question Three	36
Question Four	38
Question Five	38
IV. Findings	39
Introduction	39
Question One	39
Introduction	39
Software Development Phases	40
Lines of code	42
Requirements Definition	44
Complexity	48
Work Breakdown Structure (WBS)	49
Amount of Prior Planning Performed	52
Software Development Standards	53
Use of Management Principles	54
Software Programmer Ability	56
Data Base Requirements	58
Allowance for Testing	58
Use of Software Development Tools	59
Identification of Resource Requirements	60
Other Factors	61
Conclusion	62
Question Two	62
Review of Literature	63
Discussion of Interview Results	67
Question Three	75
Software Development Schedule Theory	75
Review of Models Selected for Analysis	77
Description of the Data	77
Analysis of Results by Model	79
COCOMO	79
PRICE-8	83
SoftCost-R	84
SPQR/20	84
System-3	85
Summary of Results	86
Question Four	87
Question Five	91
V. Conclusions and Recommendations	93
Conclusions	93
Recommendations	95
Appendix A: Interview Questionnaire For DOD Program Managers/Engineers	97
Appendix B: Interview Questionnaire For Software Development Experts (DOD and Comercial)	104

	Page
Appendix C: Mitre Project Data	105
Appendix D: Intermediate COCOMO Cost Driver Ratings And Effort Multipliers For Project #24	123
Appendix E: PRICE-S (MODE 2) Input Values	124
Appendix F: PRICE-S (MODE 2) Estimation Summary	128
Appendix G: SoftCost-R Input Values	129
Appendix H: SoftCost-R Resources Estimate	131
Appendix I: SPQR/20 Input Values	132
Appendix J: SPQR/20 Summary Estimate	134
Appendix K: System-3 Input Values	135
Appendix L: System-3 Summary Report	137
Appendix M: COCOMO Software Development Effort Multipliers	138
Bibliography	139
VITA	145

List of Figures

Figure	Page
I. Software Development Project Feedback Loop	55
II. Rayleigh-Norden Curve	76

List of Tables

Table	Page
1. COCOMO Factors By Category	19
2. PRICE-S Cost Elements and Associated Development Phases	21
3. Factors Consistently Identified By Experts as Affecting Schedule	62
4. Model Input Parameters Identified As Having The Highest Correlation With Schedule	74
5. Estimated Project Duration by Model (in months)	87

List of Equations

Equation	Page
1. Intermediate COCOMO Organic Equation For Effort	19
2. Intermediate COCOMO Semi-Detached Equation For Effort	19
3. Intermediate COCOMO Embedded Equation For Effort	19
4. Norden Equation For Cycles Of A Project	76
5. Basic COCOMO Embedded Equation For Effort	79
6. Intermediate COCOMO Adjusted Estimating Equation	80
7. Basic COCOMO Embedded Equation For Schedule	80
8. ESD Recalibrated Basic COCOMO Embedded Equation For Effort	80
9. ESD Recalibrated Basic COCOMO Embedded Equation For Schedule	80
10. ESD Recalibrated Nominal Intermediate COCOMO Equation For Effort	80
11. ESD Recalibrated Adjusted Intermediate COCOMO Equation For Schedule	80
12. PRICE-S Model Equation For Schedule	89

Abstract

Accurate schedule estimation in software development programs is important because delays in the schedule of a software development program can cause delays in the entire schedule of a weapon system.

In order to more accurately predict the schedule of a software development program, estimators need to know which development factors affect schedule. This thesis reports twelve factors identified as heavily influencing software development program schedules. These factors were determined through extensive reviews of literature written by software development experts and from interviews with DOD Program Managers/Engineers and commercial experts who have had experience with software development programs.

Also, there are many commercial software development estimation models on the market today. Five of these models were analyzed for their accuracy in predicting software development programs. The models analyzed were COCOMO, PRICE-S, SOFTCOST-R, SPQR/20, and SYSTEM-3. Inputs to these models were also analyzed for their correlation to schedule prediction. (KR) ←

AN ANALYSIS OF SCHEDULE DETERMINATION
IN SOFTWARE DEVELOPMENT PROGRAMS AND
SOFTWARE DEVELOPMENT ESTIMATION MODELS

I. INTRODUCTION

General Issue

In today's Defense world, the use of the computer is no longer a new phenomenon. The widespread use of computers in military weapon systems has made software development cost an important part of a weapon system's cost estimate. The ability to determine the cost of the software used in these computers, however, is still in a growth state. Volume I of the Doty Associates, Inc. Software Cost Estimation Study expands this belief.

Since the advent of modern computers, it has been common for the cost and time required to develop software, particularly for large programs, to exceed initial estimates. In addition, the increased sophistication of software applications over the past ten years has made these erroneous estimates more significant in terms of absolute costs (18:1).

Although this study was written in 1977, these statements still hold true today. Barry Boehm, developer of the COCOMO software development program estimation model, continues on the need for software cost estimation.

There is no good way to perform a software cost-benefit analysis, breakeven analysis, or make-or-

buy analysis without some reasonably accurate method of estimating software costs, and their sensitivity to various product, project and environmental factors [5:30].

The schedule of the software development project is one major contributor to the project's cost. Schedule risk, an aspect of scheduling, has a large impact on how much additional cost may be incurred. Alan Wingrove, in his recent article, "The Problems of Managing Software Projects," suggests schedule is even more important in larger sized software development programs when he states,

From the evidence it would appear that any project which involves a significant software content runs a high risk of being completed late and costing significantly more than budgeted [46:3].

Because of today's more advanced and complex software that is continuously being developed for national defense, schedule and schedule risk is an area that must be frequently examined in order to develop new and more precise methods for its estimation. When schedule is accurately estimated, cost estimates will improve with accuracy and overruns will be curtailed.

Definitions

Before continuing the introduction, some key terms that will be used throughout this research effort will be defined.

Computer Software Component (CSC). A CSC is a lower sub-division of Computer Software Configuration Items (CSCIs) or CSCIs that have been partitioned into smaller units (21:3).

Computer Software Configuration Item (CSCI). CSCIs are computer programs, or groups of computer programs satisfying common functions. They are managed as separate entities (21:3); each CSCI follows its own development cycle. CSCIs will be discussed later in a review of DOD-Standard (DOD-Std) 2167.

Cost Model. The AFSC Cost Estimating Handbook defines cost model as "an estimating tool consisting of one or more cost estimating relationships, estimating methodologies, or estimating techniques used to predict the cost of a system..." (41:8-2).

Milestone. John Boddie, author of Crunch Mode: Building Effective Systems on a Tight Schedule, states, "A milestone event is an event that must occur if the project is to be completed. The combination of the event and the date it should occur comprise a milestone (3:58).

Schedule. A "timed plan" for completing a work package (12:146). In his article, "Managing Software Development Projects for Maximum Productivity," Norman R. Howes states:

The purpose of scheduling is not only to predict when a job can be completed given the sequence of work and the resources available, but also to establish start and end dates for each work package [23:29].

Schedule Risk. Schedule risk is the probability of a software development program not being completed within the time frame for which it has been budgeted (41:A-65).

Software Development Project/Program. A software development program or project is the process of engineering software to be used in conjunction with some type of hardware. Paul Rook continues this definition in the article, "Controlling Software Projects."

The clear emphasis in the modern approach to software engineering is to focus attention on the overall development process. This is the aim of structured software development, which breaks down the project into a series of distinct phases, each with well defined goals, the achievement of which can be verified, ensuring a sound foundation for the succeeding phase [37:7].

And finally, as another example, Norden defines a development project as "a finite sequence of purposeful, temporally ordered activities, operating on a homogeneous set of problem elements, to meet a specified set of objectives . . ." (27:74).

Virtual Machine. Boehm states that "for a given software product, virtual machine is the complex of hardware, and software it [the computer] calls upon to accomplish its task (6:510).

Specific Problem

Because delays in schedule can have a substantial adverse affect on costs, program managers need to know how and to what extent schedule risk should be included in software development cost estimates.

Justification

The Cost Analysis Branch of the Aeronautical Systems Division (ASD/ACC) of Air Force Systems Command at Wright-Patterson Air Force Base has requested that AFIT thesis research be done in the area of software development cost estimating (43). Schedule risk plays an important part in accurately estimating the cost of a software development program. Determining how and to what extent schedule risk should be included in software development cost estimates will increase the accuracy of the estimates and allow weapon system program managers to better budget their resources.

Assumptions

The following assumptions are made for this research:

1) The manner in which schedule risk is incorporated into current software development cost models can be inferred from current literature and software development cost model documentation.

2) Historical data on estimated versus actual costs of software development programs can be obtained from Electronic Systems Division (ESD), Cost Analysis Branch (ACCR).

3) Cases of software development programs at ESD where delays in schedule were reported are available for review. Cases of software development programs which were on or ahead of schedule will also be available for review.

4) Software development cost analysts, determined to be experts in the field, interviewed throughout the Air Force

and in private industry, have sufficient knowledge to answer questions regarding the determination of schedule risk in software development cost estimates.

Research Objectives

In order to thoroughly investigate and derive a significant conclusion to the research problem, the following research objectives were achieved:

1) Gained sufficient knowledge in the subject of software development programs to understand the estimation process. Particularly, gain knowledge in the area of schedule and schedule risk in software development programs to be able to determine improved methods for determining it;

2) Developed adequate expertise in using the designated cost models to be analyzed so that the extent to which the models incorporate schedule risk can be assessed;

3) Determined the relative importance of schedule risk with respect to other cost drivers in software development cost models;

4) Determined to what extent schedule risk has been previously, and currently should be incorporated into cost models based on its determined importance.

5) Developed criteria, based on this research, that program managers can use when determining how schedule risk can more accurately be incorporated into a cost model.

Investigative Questions

The following investigative questions are raised to support the research objectives:

- 1) What are the factors affecting a software development program schedule?
- 2) What is the importance of schedule risk to a software development program?
 - a) To what degree do program managers consider schedule and schedule risk?
- 3) What methods for determining schedule risk are currently used in software development cost models and have they been tested and/or validated?
 - a) Which of these methods appear to be most valid?
- 4) In cost models, what is the significance of schedule risk?
 - a) Is schedule risk an independent variable or does its significance depend on the value of other independent variables in the cost models such as size of the program, number of programmers required, or level of software sophistication required?
- 5) Can any of these methods be combined or incorporated into a new and more accurate method for accounting for schedule risk in cost models?

Scope and Limitations

The following limitations will define the scope of this research effort:

- 1) Only four or five current software development cost models were chosen for analysis.
- 2) The selection of the cost models depended on the availability of thorough documentation for each model, the availability of the model itself and the ability to access these models to run case study data.

3) Only one project from the data base was run for analysis on each of the models.

4) Interviews with program managers who have been involved in software development programs were arranged through points of contact at various product divisions throughout Air Force Systems Command and other USAF major commands.

II. Literature Review

The process of software development is a very popular topic among a broad base of people ranging from specialists in computer programming, who are interested in all the intricacies of programing new software, to general program managers who are only interested in the end result. Consequently, there has been a great deal of literature written on software development ranging from very technical to very general. However, only a small portion of this literature specifically addresses schedule and schedule risk as a component in software development cost estimating.

The purpose of this chapter will be to address current views on software development cost estimating and, specifically, what are considered to be some of the determinants of schedule. A review of the current Department of Defense Standard directing the development of software will be given, and cost models that will be used in this research effort will be discussed.

Software Development

The increase in size of many current software development projects and the common occurrence of overrunning schedule with its associated increased costs has caused many major corporations and the DOD to examine ways of improving their software development techniques. A representative from

TRW, Inc. gives three reasons why examining software development more closely is important.

1) Our customers have shown a growing unwillingness to accept cost and schedule overruns unless the penalties were increasingly borne by the software developer.

2) Partly as a consequence of 1), we have entered into software development contracts where both adherence to predicted costs and on-time delivery were incentivized. That means we made more money if we could predict accurately the cost and the time it would take to do the job.

3) We found that we could improve our estimates only by improving our understanding of exactly what steps and processes were involved in software development, and this understanding enabled us to manage the effort better. The better management, in turn, improved our estimates [47:156].

For these reasons, it is also important that personnel in the DOD examine the processes that comprise software development and understand what occurs at each phase.

As noted above, software development is a multi-faceted development process. It is best managed and understood when broken down into distinct phases. Norman R. Howes, author of "Managing Software Development Projects for Maximum Productivity," gives an interesting perspective on the software development process. He theorizes that software development management has two distinct parts: project planning and project execution (23:27). He also says that these two parts each contain several distinct sub-parts. With respect to schedule the main part that should be analyzed, according to Howes, is "project planning." Howes lists five sub-parts to project planning: subdivision of

work, quantification, sequencing of work, budgeting, and finally scheduling (23:27).

Normally, software development is divided into subsections for different phases of the development. An example of division is the Brown & Root Integrated Control System, which Howes helped in creating, a development management system that divides software development into a "series of decompositions based on how the work will be performed" (23:27). The resultant hierarchy is called a "work breakdown structure" (WBS) (23:27). The concept is often used by other software development teams to better control all aspects of the project.

Jack Cooper explains basically the same idea for software development schedules. He states that an effective way to develop a schedule is to "approach the task the same way you would approach the top-down design of a software system" (14:24). He says that when developing the schedule, start at the top of the project and "decompose it into its first line major tasks" (14:24). Next, the task can be further decomposed into more comprehensive sub-tasks until "the level is reached where tasks cannot be sub-divided any further" (14:24).

Determinants of Schedule Risk

Another area of interest to this research effort is the determinants of schedule risk as perceived by experts. These determinants are important to any software development

project and can vary widely depending on expert opinion and the particular case to which the schedule risk applies. Cooper in the article "Software Development Management Planning" reiterates this point, "Another critical action to be taken before proceeding . . . is to identify all of the potential risk areas" (14:23). He goes on to state, "Many of the high risk areas should be included on the project's critical path" (14:23). While all determinants are important, this review will focus on specification, experience, planning, and complexity.

One of the major determinants of schedule risk has to do with specification. If the requirements for the software are not properly specified and defined, accurate schedule determination will be difficult. Walt Scacchi, in his article "Managing Software Engineering Projects: A Social Analysis," talks about specification; he states, "Problems found in specifications may be due to oversights in their preparation or conflicts between participants over how they believe the system should function" (38:54).

Another factor that can determine schedule risk is experience, specifically, experience that the software developers have with the type of software being developed. The Doty Associates, Inc. report highlights this area when analyzing sizing of the program and experience of the developer at the same time. If the developer determines size estimates in man months and secondary resources based on

experience, then the chance for error will depend on the similarity of his/her previous experience to the new development (18:72). The sizing of the software development program will in turn determine its schedule.

The level of planning involved is also a determinant of schedule risk. Generally, the more planning that goes into a software development project before it starts, the greater the chances it will run on schedule. According to R.S. Hurst in the article "SPMMS-Information Structures In Software Management,"

Planning includes planning the project, planning the product and planning the use of resources; it includes choosing the processes to be applied within the project and determining the nature of the support the project will need. A plan has to describe the relationships between the tasks, the intermediate outputs and the responsibilities of personnel (24:50).

One last major determinant of schedule risk discussed here is complexity. The more complex or sophisticated the software that is being developed, the higher the probability there will be delays or deviations in schedule. In one study of product-related factors on productivity, it was found that a large percentage of complex code was associated with low productivity (44:146). Low productivity is often associated with schedule delay.

DOD Standard 2167

DOD Standard (DOD-STD) 2167, "Defense System Software Development," is the Department of Defense (DOD) standard

which establishes "requirements to be applied during the acquisition, development, and support of software systems (16:1/2). DOD-STD 2167 is the result of a development process started in 1979 to standardize acquisition, development, and support standards and policies (11:1). Prior to DOD-STD 2167, DOD-STD's 483, 490, 1521A, and 1679 provided the guidelines for software development (11:1). Cheadle believes the new standard impacts software parametric modeling "because it requires specific documentation to be reviewed at specified design reviews (11:1).

One important aspect of DOD-STD 2167 is that it breaks the software development process into the following major activities, which it says may overlap or be applied iteratively:

- 1) System Requirements Analysis/Design
- 2) Software Requirements Analysis
- 3) Preliminary Design
- 4) Detailed Design
- 5) Coding and Computer Software Unit Testing
- 6) CSC Integration and Testing
- 7) CSCI Testing
- 8) System Integration and Testing (16:9).

DOD-STD 2167 also calls for formal reviews and audits to be conducted at specified points during these software development activities. Between System Requirements Analysis and System Design is the System Requirements Review (SRR);

between System Design and Software Requirements Analysis is the System Design Review (SDR); between Software Requirements Analysis and Preliminary Design is the Software Specification Review (SSR); between Preliminary Design and Detailed Design is the Preliminary Design Review (PDR); between Detailed Design and Coding and CSU testing is the Critical Design Review (CDR); between CSC Integration and Testing and CSCI Testing is the Test Readiness Review (TRR) (16:10). After CSCI Testing, three more reviews occur before Testing and Evaluation and finally Production and Deployment. These reviews are Functional Configuration Audit (FCA), Physical Configuration Audit (PCA), and Formal Qualification Review (FQR) (16:30).

Cheadle believes the additional regimentation of the review process and the introduction of the new Software Specification Design Review (the SSR) was initiated "because contractors and contracting agencies were still discussing requirements at the Critical Design Review" (11:1). Cheadle states, "If the contractor, user and customer are still identifying requirements at the CDR then the project is in danger of overrunning and missing the schedule" (11:1).

Bruce and Pederson state during the Preliminary Design Phase, "requirements analysis tasks are performed to establish a requirements baseline" (8:8). They go on to state "the requirements are then analyzed and allocated to functional software areas which results in a preliminary

design (8:8). This preliminary design will provide the baseline for the Detailed Design phase (8:8).

Next, further analysis and design work on the preliminary design baseline results in the detailed design, which forms the baseline for Coding and CSCI Testing (8:8). During this phase the actual coding and testing activities occur (8:8).

This new structure seems to be an attempt by the DOD to standardize the phases of the software development process. However, often, many experts in software development have different definitions for activities of a software development project from those of DOD-Std 2167. Bruce and Pederson state that software development proceeds through three distinct phases: Preliminary Design, Detailed Design and Implementation and Operation (8:8). They also note that it is often very difficult to determine the actual status of any of these development activities (8:8). Bruce and Pederson state, "Often five or six development steps are completed and three fourths of the calendar time and budget is expended before any proof of progress or quality is shown . . ." (8:8).

Software Development Estimation Models

This research will be conducted with the aid of commercially and DOD-developed software development cost models. There are many of these types of cost models available for use today. The cost models to be used in this

analysis will be determined by the extent to which thorough documentation is available on the model. The following paragraphs describe cost models that are candidates for use in this research effort.

The Constructive Cost Model (COCOMO). COCOMO is one of the more popular software cost estimating models on the market today. This is probably true because its developer, Barry W. Boehm, provides extensive documentation as to how it was developed and how it works. Users can make adjustments to the model to fit their own scenarios.

Boehm emphasizes the primary reason he developed the COCOMO model was to help managers understand "the cost consequences of the decisions they will make in commissioning, developing, and supporting a software product" (4:13). Boehm describes the different COCOMO models.

COCOMO is actually a hierarchy of three increasingly detailed models which range from a single macro-estimation scaling model as a function of product size to a micro-estimation model with a three level work breakdown structure and a set of phase-sensitive multipliers for each cost driver attribute (4:13).

In this research effort, the Intermediate COCOMO model (the second of the three models described above) was analyzed for its techniques in incorporating schedule. The Intermediate COCOMO model is an extension of the Basic COCOMO model (described above as a single macro-estimation scaling model). The Basic COCOMO model is good for quick, early rough order of magnitude estimates of software costs, but its

accuracy is limited because of the lack of additional factors used to compute the estimates (4:58). The Intermediate COCOMO model has potential for greater accuracy and a higher level of detail, making it more suitable for cost estimation in the more detailed aspects of software product development (4:58).

Boehm stated, "There are many candidate factors to consider in developing a better model for estimating the cost of a software project" (5:115). To narrow this large number down to a manageable size Boehm subjected each factor to two tests:

- 1) General Significance--The test for general significance eliminates those factors significant only on a small number of specialized occasions (5:115).

- 2) Independence--The test for independence eliminates factors strongly correlated with product size and compresses factors usually highly correlated on size projects into a single factor (5:115).

The result of this narrowing of the number of factors is the Intermediate COCOMO currently uses 16 different cost drivers which are divided into four categories to estimate cost (21:9). These categories are Product Attribute, Computer Attributes, Personnel Attributes and Project Attributes (5:116). The factors for each category are listed in Table 1.

Table 1

COCOMO Factors By Category (5:115-116)

Product Attributes

Software Reliability (RELY)
Data Base Size (DATA)
Product Complexity (CPLX)
Requirements Volatility (RVOL)

Computer Attributes

Execution Time Constraint (TIME)
Main Storage Constraint (STOR)
Virtual Machine Volatility (VIRT)
Computer Turnaround Time (TURN)

Personnel Attributes

Analyst Capability (ACAP)
Applications Experience (AEXP)
Programmer Capability (PCAP)
Virtual Machine Experience (VEXP)
Programming Language Experience (LEXP)

Project Attributes

Modern Programming Practices (MODP)
Use of Software Tools (TOOL)
Required Development Schedule (SCHED)

Note: Requirements Volatility was added in 1986 (21:9).

The Intermediate COCOMO software development effort first begins by generating nominal effort from scaling equations (5:117). The equations for the various types of software are as follows:

Organic	$MM_{nom} = 3.2 (KDSI)^{1.08}$	(1)
Semi-detached	$MM_{nom} = 3.0 (KDSI)^{1.12}$	(2)
Embedded	$MM_{nom} = 2.8 (KDSI)^{1.20}$	(3)

where,

MM is man-months,

KDSI is thousands of deliverable source instructions.

These nominal estimates are then adjusted using ratings with respect to the other 16 cost driver attributes described above (5:117). The COCOMO model also uses the Rayleigh distribution to give approximations to the labor distributions for the software development effort (5:68).

The output of the Intermediate COCOMO model is the level of effort in person-months (5:115). A COCOMO person-month consists of 152 hours of working time "which was found to be consistent with practical experience with the average monthly time off due to holidays, vacations and sick leave" (5:59). COCOMO estimates in person-months instead of dollars because of "the large variations between organizations in what is included in labor costs . . ." (5:59).

PRICE-S. PRICE-S™ is a commercially available (GE, Inc.) macro-cost estimation model developed primarily for embedded system applications. The model consists of parametric methods to estimate costs and manpower for software development (41:8-21). The AFSC Cost Estimating Handbook further describes PRICE-S.

PRICE-S estimates probable cost on the basis of project scope, program composition, processor loading, and previous organizational performance. Operational and testing requirements are incorporated, as well as technology, growth and inflation, to generate estimates of cost . . . (41:8-21).

In addition to cost, PRICE-S will derive schedules of work. To do this, the model examines schedule constraints that have been imposed within the model (34:I-2). Costs are

also adjusted to account for acceleration, stretch-out, and phase transition inefficiencies (34:I-2).

The basis for PRICE-S is a comparison of new requirements to analogous estimates in the past (34:I-2). The model relies heavily on the experience and judgement of managers using the model. PRICE-S incorporates this experience into variables which "describe the significant technological and cost differences between individual projects and organizations (34:I-2).

PRICE-S outputs values for six cost categories in each of the nine development phases as prescribed by DOD-STD 2167. Table 2 displays the cost elements and the corresponding development phases as described in the PRICE-S manual (34:I-1):

Table 2

PRICE-S Cost Elements and Associated Development Phases	
<u>Cost Element</u>	<u>Development Phases</u>
Design	System Concept
Programming	System/Software Requirements
Data	Software Requirements
System Engineering/ Program Management	Preliminary Design Detail Design
Quality Assurance	CSC Code/Unit Test
Configuration Management	CSCI Test System Test and Evaluation Operational Test & Evaluation

The PRICE-S model calculates cost in terms of effort (either person months or person hours) and a typical schedule for the development program (34:I-1). PRICE-S will also perform sensitivity analyses and summarize effects of uncertainties (34:I-3).

PRICE-S inputs are grouped into nine categories as follows:

- 1) Project Magnitude--the number of source lines of code (SLOC) to be produced.
- 2) Project Application--called Application, tells what type of project.
- 3) Level of New Design and Code--amounts are specified by the user.
- 4) Productivity--entails organizational capabilities, experience and individual talents of the activity that will accomplish the work.
- 5) Utilization--effort required to fit a software program into a processor.
- 6) Customer Specifications and Reliability Requirements--called Platform, summarizes operational requirements. Also used to describe the transportability requirements of a software project or how often a program will be moved from one type of hardware to another.
- 7) Development Environment--called Complexity, describes the effects of environmental factors that can directly affect schedule time.

8) Technology Growth--based on the possibility of using new, innovative development techniques that make the development process more efficient. The key drivers here are Productivity Factor, Application and Time.

9) System Integration--factor based on the necessity to merge two or more related software products into one system. PRICE-S will develop time and schedule estimates for this activity the same as it does for individual sub-systems (34:I-11).

Regarding schedule, PRICE-S basically takes the judgement of experienced managers, engineers and estimators to determine the impacts of the key cost drivers and incorporates this knowledge into the model (34:I-12). As with any values based on expert judgement, these values would be subjective. However, the PRICE-S Manual states that as much as possible, "actual recorded data is used to formulate, test, and verify those assessment processes" (34:I-12). The PRICE-S Manual also acknowledges that data does not always exist. The manual gives the example that "the impact of schedule variations on cost cannot be statistically processed" (34:I-12). Since there was only one schedule for programs in the past, it is not certain what would of happened had that schedule been shorten or lengthened. PRICE-S contends, however, that by knowing actual schedules differ from the original planned schedule, cost impacts can

be modeled through studying the processes employed to manage the schedule (34:I-12).

PRICE-S uses the planned completion date for Software Specification Review (SSR) and the Complexity factor to generate an "internal reference schedule" which is used to calculate effort penalties (34:I-B).

When additional dates are entered (other than SSR), new schedule dates are calculated to meet schedule constraints and they are compared with the internally calculated reference schedule. This comparison is used to calculate effort penalties associated with phase acceleration, stretch-out, and deviations from reference schedule [34:I-B8].

PRICE-S outputs cost in person-months or hours by the software the life cycle phases listed in DOD-STD 2167. It will also list schedule information by review milestone (e.g. SRR, SDR, etc.) (34:I-3).

SLIM. The Putnam Software Lifecycle Model (SLIM™) model is a software cost estimating system available from Quantitative Software, Inc. (31:1-1). The model is based on much of the theory developed by Mr. Lawrence Putnam (31:1-1). SLIM is a fully interactive model and is used to generate projections of cost, time, personnel and machine resources for developing computer software systems. It is designed to handle a front-end estimating problem because it requires certain estimate information from the start (31:2-3).

Inputs for SLIM consist primarily of three SLOC

estimates: minimum, most likely, and maximum (21:14). The following other inputs are also required:

- 1) Language
- 2) System Type
- 3) Description
- 4) Percentage of hardware memory used
- 5) Experience
- 6) Modern Practices (percentage use of new development techniques)
- 7) Technology Factor (measure of difficulty)
- 8) Other Factors (including labor rates and economic factors) (21:14).

The AFSC Cost Estimating Handbook describes SLIM outputs.

The model provides the following outputs:

Identification of minimum cost, minimum time, and all feasible solutions for a particular software development project

Estimates of monthly man-loading

Optimum schedule for completion with associated milestones

Risk profiles for schedule and effort

Identification of constraints on manpower application and completion schedules [41:8-24].

As noted above, SLIM gives the minimum feasible time.

The user may then use this time schedule or he may specify a longer time in which he can take advantage of the trade-off law. This law is in essence a quantification of the Brooks' trade-off law which states that one can greatly reduce the

cost and effort by taking a little more time (31:1-1).

SLIM has a "design-to-cost" function which will generate feasible time schedules given user-specified constraints. SLIM will check user-specified cost and time inputs for feasibility and consistency with past data (31:1-1).

SLIM also has an optional life cycle output that will derive person-months, schedule, and person loading profiles (21:15). William G. Cheadle stated that SLIM gives the shortest schedule first and then "it implies you can save money by moving the time out to the optimum schedule" (10).

SLIM relies heavily on the Rayleigh-Norden Curve to allocate resources during a project. The manufacturers of the SLIM model contend that the approach used for a software estimating problem depends on where one is in the software life cycle (31:2-3). They further contend that the software development program problem is a "pure estimating problem" during the feasibility and function design phases (31:2-3). It is pure estimating because the problem uses phenomenology and past experience (data) to forecast a future event (31:2-3). The developers of SLIM found that in this scenario a model of observed behavior would be appropriate. They also wanted a model that allowed the time to vary and had input parameters of development time, development effort and cost (31:2-3). For these reasons they chose the Rayleigh-Norden curve as the basis for their model (31:2-3).

SoftCost-R. SoftCost-R™ is based on the model Dr. Robert Tausworthe of the Jet Propulsion Laboratory developed for NASA in 1981 (42:1-2). This model is also based on the Rayleigh-Norden curve as well as Putnam theory developed in SLIM (42:1-2).

SoftCost-R bases its estimates from inputs from the factors size, management, staffing, complexity and environment which is input at the beginning of the model operation (42:1-2). From this information, SoftCost-R computes a resource estimate in three steps:

- 1) Size in Kilo Source Lines of Executable Code (KSLEC) is computed.
- 2) Productivity as a function of technological and environmental factors is computed.
- 3) Effort is computed by dividing total size by productivity (42:1-2).

A standard WBS is also used to produce the task plan and schedule to be used during initial project planning stages (42:1-2). SoftCost-R also incorporates a version of COCOMO (COCOMO-R) in conjunction with SoftCost-R into the estimate as a sanity and reasonableness check (42:1-3).

Outputs of SoftCost-R include an optimal time solution in terms of time, cost and effort, and the statistical confidence associated with the estimate (21:16). The estimate of the resources required to complete the project is

defined in terms of the project factor data (listed above) (42:3-22).

Regarding schedule, initial inputs into the SoftCost-R model determine how SoftCost-R will determine schedule. "Estimate Date" and "Project Start Date" are two inputs used to compile Gantt and PERT charts (42:1-3). Once the option of either Gantt or PERT chart is selected, the model will display effort and duration values originally input at the beginning of the program and then give the user the opportunity to change these values if further knowledge is available (42:3-30). The Gantt and PERT charts are geared to DOD-STD 2167. Another option, "what-if" has the capability of displaying the effects of varying the schedule on a given budget and vice versa (42:3-24).

SPQR/20. The Software Productivity, Quality, and Reliability Estimator (SPQR/20™) was developed in 1986 by Software Productivity Research, Inc. (39:1). SPQR/20 is designed to be the quick estimator version of this model. The number "20" represents the approximate number of input variables required for the model's predictions (39:2).

Features of SPQR/20 include prediction of schedule by phase, effort and costs by activity, and staff sizes (39:1). SPQR/20 will also predict complete development cycles from planning through delivery, maintenance and enhancements for five years after delivery, defect levels of software projects, defect removal efficiencies of reviews and tests,

and the quality and reliability of the delivered software (39:1).

SPQR/20 is designed to estimate "all of the direct labor applied to software development and maintenance" (39:2). The following are the major activities included in SPQR/20 estimates:

- 1) Planning
- 2) Requirements
- 3) Design
- 4) Coding
- 5) Integration
- 6) Testing
- 7) Documentation
- 8) Management
- 9) Central maintenance
- 10) Enhancements (39:3).

Another interesting aspect is SPQR/20 uses "function points" to predict new source code size. The Function Point technique was developed in 1979 by A.J. Albrecht of the IBM Corporation (39:50).

Prior to the Function Point technique, software productivity was always measured in terms of lines of code such as cost per source line or lines of code per man month. Unfortunately, this metric cannot safely be used for high-level languages, since productivity rates in lines-of-code form actually move backward as real productivity improves (39:50).

In other words, there are some non-coding efforts that will remain as fixed costs in person months despite the use

of a high-level language that reduces the amount of effort for coding. The non-coding efforts are requirements, design, documentation, and management. Lines of source code, coding, and integration and test are the coding efforts affected by the use of a high-level language (39:50). As in any process where there are fixed costs involved, when there is a decline in the number of units produced (or in this case a reduction in lines of code due to use of a high-level language), the cost per unit (or source code) must increase.

The Function Point technique attempts to compensate for the use of high-level languages by placing weights on parameters that Albrecht determined to embody the functionality of a program. These parameters are: number of inputs, number of outputs, number of inquiries, number of data files, number of interfaces (39:51). When the parameters are weighted, they are also adjusted for complexity and then summed to derive a function point total (39:51). This Function Point value is input into the model to estimate new source lines of code.

The advantages of using the Function Point technique as noted by the developers of SPQR/20 are:

- 1) Function Points are independent of source code, and do not penalized high-level languages;
- 2) Function Points can be applied early in a software life cycle, such as during the design phase;
- 3) Function Points can be used to predict source code size . . . [39:51].

The disadvantages of the Function Point technique lie in the ambiguity that exists in defining the Function Point

parameters and the subjectiveness in the treatment of the complexity adjustment (39:51).

Output of SPQR/20 covers six aspects of software development programs. The first output is a risk and quality estimate. Next, is a defect removal and reliability estimate. Next, are the main development and maintenance cost estimates. Finally, normalized management information is output.

System-3. System-3™ is a software cost estimating model developed by Computer Economics, Inc. and is based on work done by Dr. Randall Jensen (13:1-1). Dr. Jensen explains the basic equation used in System-3 in the article "An improved Macrolevel Software Development Resource Estimation Model". Using a technology constant based on technology input parameters and the Rayleigh-Norden curve, System-3 computes the required software development effort in staff-months and dollars (25:1).

JS-1 and JS-2 were the predecessors of System-3. JS-1 was introduced in 1982 after 3 years of development by CEI (13:1-3). JS-2, introduced in 1984, was a refinement of JS-1 and had many advantages over JS-1. The JS-1 produced estimates for the Development Phase only but JS-2 also produced estimates for the Requirements and System Integration Phases (13:1-4). Each parameter is further estimated at its minimum, most likely and maximum (13:1-4). CEI purports that these improvements allow users to estimate

even their uncertainty and further increase overall estimation accuracy (13:1-4).

Another feature of JS-2 was the analysis it provides for the cost and schedule required to "change, enhance and modify" pre-existing software as opposed to re-building all new (13:1-5).

Finally, JS-2 determined cost and schedule risk for different bidding situations from fixed price bids (where a higher probability of completion is required) to simpler situations where "most likely" estimates are needed. These features were firsts in parametric estimating (13:1-5).

In the spring of 1986, CEI replaced JS-2 with a new, further improved product, the System-3 (13:1-5). Some of the key estimates of the new System-3 include: minimum development time, minimum cost within a schedule, staffing projections and plans, operational support costs, project level cost summaries, software to software estimation, incremental development estimation, system conversion estimation and risk evaluation and reduction (13:1-7). System-3 will also generate reports on schedule risk, cost risk, dollars by month and differences from baseline. Finally, System-3 can generate graphs on such areas as risk analysis and effort versus schedule tradeoffs (13:1-7).

Inputs into System-3 fall under the following four basic parameter categories: size or source lines of code (SLOC), complexity, development capability, and environment (13:1-11). Minimum, most likely, and maximum values must be input

for each factor. System-3 also requires SLOC inputs (21:18).

Regarding schedule, System-3 contains a "view window" which shows what effect a change in an input parameter will have on development cost and schedule (21:18).

Outputs of System-3 consist of summary reports of effort (in dollars and staff-months). Included in the reports are development time, the computed technology constant, and the effective size (21:18).

III. METHODOLOGY

Introduction

The purpose of this chapter is to describe the methodology that will be used to arrive at answers or conclusions for the Investigative Questions posed in Chapter I. The Investigative Questions are designed to go from the general to the specific.

Question One

What are the factors affecting a software development program schedule?

This question will be answered in terms of the views experts in the field of software development and software engineering have toward schedule risk. An understanding of these views regarding schedule risk will be obtained from reading current literature in the form of books, periodicals, and professional journals on the subject. A particular journal, the Institute of Electronic and Electrical Engineers (IEEE) Transactions on Software Engineering, has numerous articles written by experts in the area of software development and is published monthly.

Question Two

What is the importance of schedule risk to a software development program?

- a) To what degree do program managers consider schedule risk when estimating the cost of a software development program?

The first part of this question will also be determined after a review of current literature written by experts in the field.

The sub-part to this question will be determined after interviewing DOD program managers and experts in private industry who have been involved in software development projects.

An interview will be used as opposed to a survey because of the "depth and detail of the information that can be secured" (20:160). Also the quality of the information should be better than if obtained by a survey because the interviewer "can note conditions of the interview, probe with additional questions, and gather supplemental information through observation" (20:160).

Costliness is one disadvantage of interviews (20:161). Costliness of interviews will not be a factor in this research effort because of the ease of reaching program managers located on the same base (Wright-Patterson AFB). Reaching program managers at ESD and other divisions of AFSC will also not be costly because they will be interviewed via Autovon.

When interviewing program managers at divisions other than ASD located at Wright-Patterson over the telephone, one possible limitation should be considered. The length of the interviews may not be as long as those conducted at ASD

interviews. This was considered when evaluating the answers given by interviewees.

The number and identity of the interviewees will be determined by contacts from the various product divisions in Air Force Systems Command. Preferably, the number to be interviewed should be at least five program managers from each Division (total of twenty program managers). This number should be sufficient to cover the spectrum of views program managers have regarding schedule risk in software development programs without the answers becoming repetitive.

The questions ranged from the general to the specific and will be open-ended type questions. Two questionnaires were used. One was for DOD Program Managers and Engineers and the other was for Software Development Experts (DOD and Commercial). The exact list of questions was determined after a review of literature was made. Once the questions were determined, they were reviewed by selected AFIT faculty and colleagues for clarity and content. The interviewer practiced asking the questions on colleagues before the actual interviews in order to become familiar with the questions. The questions used in the interviews are given in Appendix A and B.

Question Three

What methods for determining schedule risk are currently used in software development cost models and have they been tested and/or validated?

a) Which of these methods appear to be most valid?

The answer to these investigative questions was determined from an analysis of selected standard software development cost models.

The selection of these cost models was based on the availability of documentation on exactly how these models were derived and availability of access to these models for generating case estimates. Models considered for use were:

- 1) Constructive COSt Model (COCOMO) (5:29),
- 2) PRICE-S (34:1-1),
- 3) Putnam-SLIM (31:1-1).
- 4) Systems-3 (13:1-1).
- 5) SoftCost-R (42:1-1).
- 6) SPQR-20 (39:1-1).

These cost models were discussed previously in the Background section of Chapter I and extensively in Chapter II.

The analysis of these cost models was sufficient to determine how schedule risk was incorporated into these models and what comprises schedule risk.

Historical cost data on software development programs conducted in the DOD was run on the models and a comparative analysis of the estimates given by each model was made. Also, each estimate was compared to the actual cost of the software development program that generated the data to determine their accuracy.

Question Four

In cost models, what is the significance of schedule risk?

- a) Is schedule risk an independent variable or does its significance depend on the value of other independent variables in the cost models such as size of the program, number of programmers required, or level of software sophistication required?

The answer to this Investigative Question was also determined by reviewing the answers given by program managers and experts in the field to the questionnaire and by analyzing the selected cost models and reviewing the available documentation on the regression techniques used to derive these models.

Question Five

Can any of these methods be combined or incorporated into a new and more accurate method for accounting for schedule risk in cost models?

The answer to this investigative question was derived from a culmination of the answers to the investigative questions above.

IV. Findings

Introduction

The purpose of this chapter is to discuss the findings of this research and to discuss the answers to each of the investigative questions posed in Chapter III. Each of the investigative questions are addressed independently; findings from this research relate to more than one investigative question.

Question One

What are the factors affecting a software development program schedule?

Introduction. Frederick P. Brooks, Jr., author of The Mythical Man-Month, states, "Most software projects have gone awry for lack of calendar time than for all other causes combined" (7:14). This regard for the effects that a schedule can have on a software development program seems to be universal among experts in the field. Therefore, it is important that the factors affecting schedule should be identified so that steps to control these factors can be taken.

Chapter III stated that the answer to this question was determined from analyzing the views of the leading experts in the field of software development estimation. The experts' views were found by researching articles and conference papers found in common professional journals and articles of

the software engineering field. Books and tutorials by noted software development experts were researched as well. The general conclusion is there are many factors that go into the estimate of a software development program, and the answer to this question can be as broad as the entire software development program estimation process itself. This section will begin with a discussion of the views of the experts examined in this research. Next, their views will be summarized, and finally conclusions will be drawn from the summary.

Software Development Phases. Before discussing the factors involved in the estimation of a software development program, a review of the broader topic of the software development life cycle is appropriate. As noted in Chapter II, DOD-STD 2167 gives a very complete description of all the phases of the software development life cycle. These phases are, in order of earliest to latest: System Requirements Analysis/Design, Software Requirements Analysis, Preliminary Design, Detailed Design, Coding and Computer Software Unit Testing, CSC Integration and Testing, CSCI Testing, and System Integration and Testing (16:9).

Many experts break down factors that are used in software development program estimation by life cycle phase and others still will group the factors by the more general headings of "product related factors" or "process related factors". J.D. Aron reports that the System Development

Corporation in the Programming Management Project spent years analyzing data to identify factors affecting a program's schedule (2:262). They concluded that key variables fall into three groups: uniqueness, development environment, and job type and difficulty (2:262).

Another expert, Stephen P. Keider, identifies projects as having five distinct phases: pre-initiation period, initiation period, project duration, project termination, and post-termination period (26:53). Keider identifies various factors that occur in each phase that can affect schedule. Still another expert, Alan J. Driscoll, describes the software development process in three stages: analysis and design, implementation, and verification (19:46). Finally, William S. Donelson has yet another idea of what the phases of the life cycle of a software development program should be. He says the life cycle has the following nine phases: problem definition, project organization, problem analysis, system definition, system review and approval, detail design, programming and testing, training and implementation, and post-implementation review (17:74-75).

The point here is that although the standard for software development programs with the DOD is DOD-STD 2167, phases of this life cycle may have other slightly different titles or certain phases may be combined into a larger singular phase in private industry. This may be confusing to some when trying to identify which phases of the life cycle

are associated with which factors that affect schedule. Also, another problem that may arise from this is that parties involved in a software development effort, because they have different definitions for the phases of the software development life cycle, may also have different ideas for where critical events in the software development life cycle should occur.

For this research effort, the life cycle phases described in DOD-STD 2167 will be used. Also, if factors are grouped in the discussion, they will be grouped by the categories identified in the COCOMO estimation model. Those categories are: size attributes, product attributes, computer attributes, personnel attributes, and project attributes (6:502). The following section discusses each factor found by various experts to affect software development program schedule.

Lines of code. As Putnam (who is well known for developing the SLIM estimation model) states, "The earliest efforts at software cost estimation arose from . . . measuring average productivity rates for workers (30:11). An estimate of the total job was made by compiling these rates. The estimate was "usually in machine language instructions" also known as "lines of code" (30:11). Machine language instructions were used in early years because the factor was related to memory capacity "which was a severe constraint with early machines (30:11). To determine the schedule of

the project, one merely took the project estimate in machine language instructions and divided it by the budgeted manpower (30:11). Putnam adds that if this method produced a schedule that was unacceptable to the user then "the manpower level (and budget) was increased until the time to do the job met the contract delivery date" (30:11).

The constraint of memory capacity is not a big problem with the computers used today for software development programs. Brooks has shown in his book, The Mythical Man-Month, that adding manpower to a software project will not decrease its schedule (especially if it is already over schedule), but it will in fact increase schedule (7:19). For these reasons, the number of lines of code a software program is expected to contain does not appear to be a good determinant by itself of what the software development program schedule will be.

Putnam adds to this conclusion by saying that most program managers do not have a good idea of how to estimate how many lines of code a program will take anyway. He says that estimating size "has largely been an intuitive process in which most estimators attempt to guess the number of modules and the number of statements per module" (32:1). Putnam contends that this may be an effective way to estimate small projects (less than 10 person years of effort), but using this method on large projects has been proven ineffective (32:1). Wendt and Evans agree with

Putnam's beliefs and say that the programmers will not be much help when estimating lines of code for a program either.

First of all, most programmers do not know how to estimate lines of code, and wonder what it has to do with anything, anyway. Programmers do not want to code the same application over and over again, and before they get to the point where they could tell you how many lines of code a particular application will take, they have requested to change areas or even moved to another company looking for a different software challenge (45:1058).

Wendt and Evans also point out that, while counting lines of code appears to not be a very efficient way of estimating the size of a software program, there is no other convenient measure available and subsequently this parameter persists (45:1058).

As a counter argument to these beliefs that the number of lines of code is not a good predictor of schedule, Roger S. Pressman, author of Software Engineering: A Practitioner's Approach, states that a program of large size could be a good predictor of a longer schedule because "as size increases, the interdependency among various elements of the software grows rapidly" and subsequently it becomes difficult to break the software down into more manageable elements (29:81).

Requirements Definition. Many experts in the field of software development contend that this factor is one of the most critical to a successful software development program which is completed on time and at target cost. Yet, Phillip

Bruce and Sam M. Pederson, authors of The Software Development Project: Planning and Management, say, "less effort is often devoted to the initial requirements definition, costing, and scheduling of a project than any other part of the development cycle" (8:16). Wingrove states that, "For anything other than a very small or simple project, this perfect requirements document is an impossibility" (46:4).

Requirements for the software program are specified by the user in the System Requirements Analysis/Design phase of the software development life cycle. At this time, the user tells the developer, as precisely as he/she can, what the software program must do. Many have found that undesirable consequences such as delays in schedule result when the user is not closely involved in the requirements definition process. Wolverton emphasizes this point when he states,

Thorough and continuous involvement of the customer in the development process has been a reality of several large software developments. Nothing takes the place of competence and communication when it comes to understanding the customer's or sponsor's requirements [47:176].

Wolverton also points out that translating total system requirements is a crucial first step in any software development project (47:176).

Driscoll explains the effect a requirements definition that is not thorough or complete can have on software development schedule.

The effects of an adequate or inadequate requirements analysis or definition ripple through all phases of software development, including design. Changes in requirements cause changes in design and these in turn usually cause schedule changes [19:47].

It is evident from this statement that it is not exactly the poor requirements definition itself that will cause a schedule to slip; instead, it is the inevitable changes that will occur later in the software development life cycle because of the poor definition that cause schedules to fall behind. Edmund B. Daly states,

. . . historical analysis of completed GTE Automatic Electric Laboratories' projects indicates that over 50 percent of all development hours are spent correcting bugs which result from faulty design [15:294].

Putnam also believes that changes in the requirements will, if not initially, eventually affect the schedule in disproportionately large amounts (33:79). He gives an example with the Army Computer Systems Command data where a program had a small change in the requirements which resulted in a major change in the schedule a year later (33:79). Putnam states, "This shows that even a modest perturbation of the system can have definite effects on large systems, and that these effects may not be apparent until a much later time" (33:79).

Bruce and Pederson emphasize that spending extra time firming up the requirements before the Detailed Design Phase can actually shorten the overall time and decrease the costs required for the software development program (7:17). They

contend that the requirements definition is "the basis for the analysis of many other costing factors, including difficulty, interfaces, size, tools, use of existing software and data base complexity" (7:17). Wendt and Evans agree with this statement when they state, "Software managers are forced to derive complete sizing of software systems based on incomplete requirements and system specifications" (45:1058).

At times, the requirements definition may be well specified in the beginning, but there are still changes in the requirements during the software development life cycle. This occurrence brings to light another factor that is related to requirements definition and will also affect schedule, which is termed as the "stability of the requirements." This factor is not discussed as frequently in literature probably because it is so closely related to requirements definition. No matter how detailed the requirements are defined, if they are not stable, changes in these requirements will result in schedule slippage. Many experts contend that if the requirements are not stable and there are major changes during the development cycle, the schedule should be discarded and a new one estimated as if a new software development program is being started.

From this discussion, it is surmised that adequately defining the requirements of a software development program is a factor that has a major affect on software development

schedule. Adequately defined requirements will allow the estimator to more accurately predict the schedule of a development program.

Complexity. A general definition for complexity is the degree of difficulty of a given kind of software routine (47:166). Complexity is typically associated with productivity in generating lines of code. It is commonly believed that the more complex a program, the more lines of code it will take and, thus, productivity will be slower. Putnam agrees that complexity has an effect on productivity. After examining very large software programs which required hundreds of lines of code to complete, he contended "it became apparent that severe departures from constant productivity rates were present and that the productivity rate was some function of the system complexity" (30:11).

Wolverton contends that determining the degree of complexity (not defining the requirements, as others have stated) of a software program is "the most crucial step in the estimating process, for it establishes the cost of the routine with all direct and indirect changes amortized against it" (47:166). In other words, knowing how complex a program will be will give a better clue as to how many lines of code the program will take. Boehm states that, "Some studies have lumped a wide variety of effects into the 'complexity' attribute and obtained relatively high productivity ranges as a result . . ." (6:505). This shows

that what determines the degree of complexity of a program is very judgmental. While it is apparent by many that complexity of a program does have an effect on that program's development productivity and subsequently its schedule, it is still difficult to pinpoint how to define a software program as being complex.

Work Breakdown Structure (WBS). Many experts believe that the existence of a Work Breakdown Structure in a software development program is vital to that program staying on schedule. Pressman states, "The degree of project structure also has an affect on estimation risk" (29:81). He refers to structure as to the ease with which a program can be broken down into a WBS or "compartmentalized" (29:81). Bruce and Pederson state the central theme of their book is that "all software projects . . . should be planned and managed along structured guidelines" (7:1). Having structure to a software development program is the primary reason many experts stress that a successful software project will have a complete WBS.

Wendt and Evans say, "Assembling the specific production tasks into a WBS is the heart of the structure which will be used for cost and schedule control (45:1060). They go on to say that because the product tasks cannot be identified in detail at the beginning of a program, the estimator must rely on the WBS to develop early estimates of cost and schedule (45:1060).

Paul Rook gives a good explanation of the importance of a WBS to a software development program:

The clear emphasis in the modern approach to software engineering is to focus attention on the overall development process. This is the aim of structured software development which breaks down the project into a series of distinct phases, each with well defined goals, the achievement of which can be verified, ensuring a sound foundation for the succeeding phase. It also breaks down the work to be performed into a series of discrete manageable packages, and creates the basis for the appropriate organisational [sic] structure. This allows overall planning of 'how' the software is going to be developed as well as considering 'what' is going to be developed as the product [37:7].

Howes also advocates decomposition of the software development into "work packages" which he says can be managed from beginning to end by one person. He says once the components are broken down to this level in the WBS, then estimation should take place for each component (23:28). Finally, Howes says, "The schedule for your project is the composite of all work package schedules" (23:29). Bruce and Pederson agree with this when they say by structuring the program, one can "reduce the estimating task to a large number of more precise estimates rather than a single task" (7:20).

Jack Cooper contends that partitioning into smaller packages allows managers to avoid "having to resort to percentages in status tracking" because partitioned packages will be small enough that "they can be considered either not started or completed (0 to 100% complete)" (14:24).

The size of these "discrete manageable packages" and adding the schedules for each package together to get a full schedule has been the subject of some discussion in literature.

Brooks has said, and many in the field seem to agree, that by continuing to partition a program into smaller and smaller components, in order to have the work progress more quickly by adding more programmers, does not always work (7:19). Brooks states,

Since software construction is inherently a systems effort - an exercise in complex interrelationships - communication effort is great, and it quickly dominates the decrease in individual task time brought about by partitioning. Adding more men then lengthens, not shortens, the schedule [7:19].

The key here is communication. Brooks believes that many tasks that are partitioned still require communication among other sub-tasks in order to be completed (7:17). This extra effort in communication more than offsets any gain in schedule that might be had by partitioning the tasks in order to add more programmers to development program.

Another point that Brooks makes is that a task by itself has a probabilistic effect on schedule (7:14). When that task is partitioned into smaller sub-tasks, each of these sub-tasks also have a probabilistic effect on schedule. When these sub-tasks are chained together to form the task, "the probability that each will go well becomes vanishingly small" (7:14).

In summary, having a WBS can help to predict the schedule of a software development program in its beginning. However, there is a point at which further partitioning of tasks in a WBS may work to lengthen a schedule rather than shorten it. This is due to the problems of communication required between the programmers and the greater probability of more errors within each task.

Amount of Prior Planning Performed. Another factor somewhat related to the WBS, and stated by many experts as having an inverse affect on software development schedules, is the amount of prior planning performed. Cooper states, "The lack of comprehensive planning prior to the initiation of a software development project is a very pervasive failing" (14:22).

A good software development plan, Cooper says, will contain "a description of the development organization, the technical approach, the milestones and schedules, and the allocation of resources" (14:22). The benefits of an adequate plan include "providing the developer with the means to coordinate schedules, control resources, initiate actions, and monitor progress of the development effort" (14:22). Rook states a good plan is based on the WBS which is produced during the work definition process (37:9).

As stated earlier, the results of inadequate planning can be disastrous. Wendt and Evans say inadequate planning can result in "a pattern of unanticipated project activities

and frequent unplanned development catastrophes and crises" (45:1055).

After analyzing the planning factor, it is evident that if the software development program has no plan or a poor plan from its onset, the estimator can assuredly add many more person-months to the schedule estimate.

Software Development Standards. Software development standards (sometimes called software metrics) are standards for development practices that have been documented as occurring on past software development projects (29:82). Many experts believe schedules can be predicted more accurately if standards that have worked in the past are used on current projects. Pressman emphasizes this belief in the following statement:

By looking back we can emulate things that worked and improve areas where problems arose. When comprehensive software metrics for past projects are available, estimates can be made with greater assurance, schedules and overall risk can be reduced [29:82].

Using standards may, in fact, improve the prediction of schedules, but the problem lies in the availability of these standards. As Wolverton states, "There are virtually no objective standards or measures by which to evaluate the progress of computer program development" (47:156). Bruce and Pederson agree when they say, "The second major problem in estimating software development costs is the lack of accurate measures of prior costs . . ." (7:17). They also believe that "without reference standards it is nearly

impossible to accurately estimate the cost of a new project" (7:17).

In conclusion, it is commonly agreed that standards, when available, will enhance the ability to accurately predict schedule.

Use of Management Principles. Another factor that experts have mentioned in literature and is related to software development standards is the use of management principles. Surprisingly, experts often mention software projects as having failed simply because there were no standard management principles or policies existing to guide the project. Thayer and Pyster commented that, in the 1960s and early 1970s, "chaos" existed in software development primarily because managers had no systematic approach available to them for managing large software projects (40:2). This time of chaos is past, and today there is a more widespread use of standard management principles for managing software development projects. Thayer and Pyster state,

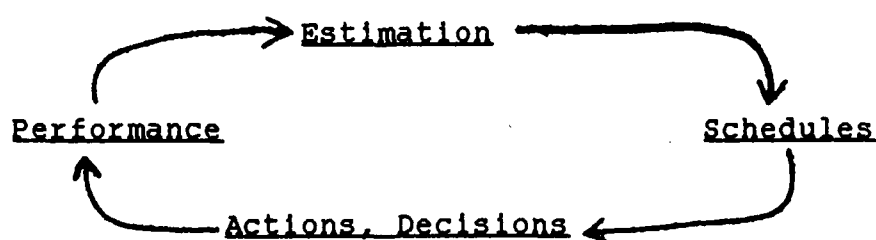
Since 1970 great strides have been made in understanding how to manage large software development projects. There have been many successful deliveries of major defense and space systems; perhaps the most well-known of the 1980's is the Space Shuttle software for NASA [40:2].

Thayer and Pyster also state that TRW, Inc. (a leader in the software development industry) requires project managers of large projects to follow a standard set of software development policies (40:2). Finally, Thayer and Pyster

comment that current advances in management science can be applied as management principles for software development projects, particularly in the area of scheduling (40:2).

T.K. Abdel-Hamid and S.E. Madnick developed a model that helps software development project managers decide when to use particular management principles and tools (1:15). Through their research, Abdel-Hamid and Madnick found that, in software development projects, a "feedback loop" similar to Figure I exists (1:19). They explain through various techniques an estimate is produced, and from this estimate a schedule is developed. This schedule is then the basis for management actions and these actions or principles used affect worker performance. Worker performance is the input for future estimates and the loop starts again (1:19).

Figure I
Software Development Project Feedback Loop (1:19)



Abdel-Hamid and Madnick's point is that "knowledge of project schedules was found to affect the real progress rate that is achieved" (1:19). They also say that all the dynamics of this loop affect the schedule of a software development program; therefore, they conclude scheduling is

not just producing better estimates but encompasses a whole host of other management problems requiring the use of management principles as well (1:19).

Wolverton says that, at TRW, they have established five management principles that "apply throughout the software development cycle to reduce the problem of control to manageable size" (47:157). These principles deal with: 1) Producing adequate software documentation that management can use to control the project; 2) Conducting technical reviews to acquire customer approval of the software criteria before a schedule is developed; 3) Controlling the software physical media to "assure use of a known configuration" throughout the development life cycle; 4) Application of software configuration management, controls and procedures; 5) Developing a data reporting, repository, and control system for use by developer and user (47:157). These principles would be a good basis for other software developers when establishing their own management principles.

Software Programmer Ability. Many experts believe that there are characteristics about the person programming the software and characteristics about the software that relate to the programmer that affect the software development program. Aron believes that such factors as programmer or programming team familiarity with the "hardware, software, and subject matter of the project" will affect the schedule of the development program (2:262). Aron also says the

development environment in which the programmers must work will affect productivity. If the environment contributes to poor communication because of dispersion of the programmers or if the facilities are unpleasant, costs will increase due to reduced productivity (2:262).

Keider says one misconception estimators have when rating the ability of a programmer at the beginning of a software development program is that programmers are considered "universally expert" and to be equally competent "analysts, designers, programmers, librarians, and documentation specialists" (26:55). Program Managers will assign any of the functions to the programmer with little regard for the programmer's ability and "invariably, this results in project delay" (26:55).

Boehm circumvents this rating of the individual programmer by saying, "the important attribute to rate is not average individual . . . programmer capability, but the effective . . . programmer team capability" (6:509). He prefers also to include in the rating such non-tangible factors as "team's cohesiveness, communicativeness, and motivation toward group versus individual achievement" in the programmer rating (6:509).

Statements made by experts in the software development field suggest that accurately rating the programmer and programmer team is important when attempting to accurately predict the schedule of a program. The problem occurs when

trying to quantify factors such as motivation that are difficult to identify on the surface.

Data Base Requirements. Data base requirements occur as a result of the outputs (reports, cathode ray tube (CRT) screens, audio response, graphics, etc.) the user requires of the software development program. As Donelson states once the user has defined the reporting requirements, "the systems analyst must then determine the data base requirement to support this reporting capability" (17:73). Donelson also urges the user to "participate aggressively in this data base definition activity" to ensure an acceptable product (17:73).

Boehm says the factor to be considered is really the "overall size of the data base to be designed, assembled, and validated prior to acceptance" (6:502). He also states,

Relatively little has been determined about the effect of this factor. The Doty study indicated it had a 'minor' effect, but no quantitative data were given. The Air Force Industry Software Cost Estimation Workshop considered it an important factor but provided no estimates on the magnitude of its effect [6:504].

From a review of the literature it is evident that several experts see the data base requirement as affecting the software development effort, but, as of yet, no conclusive evidence exists to show the extent of the effects.

Allowance for Testing. This factor describes the amount of time allowed in the life cycle for System Integration and Testing. Ideally, if the software program is developed

correctly and perfectly, the amount of time required for testing and debugging should be zero; however, this is never the case. Brooks elaborates,

No parts of the schedule are so thoroughly affected by sequential constraints as component debugging and system test. Furthermore, the time required depends on the number and subtlety of the errors encountered. . . . Because of optimism, we usually expect the number of bugs to be smaller than it turns out to be. Therefore, testing is usually the most mis-scheduled part of programming [7:19].

Donald J. Reifer, developer of the SoftCost-R software cost estimation model, says that one component of a sound technical approach to software development is that adequate attention be placed on testing (35:126).

There are tools currently available to assist in detecting errors in software and assist in making the test phase of the life cycle as efficient as possible. However, as Wingrove notes, "Reports of a lack of discipline on test methods and metrics can have catastrophic consequences for interfacing and integration" (46:5). This in turn could lead to further schedule delay.

The main point regarding test is that it should not be optimistically shortened when developing the development program schedule; but instead, it should be realistically lengthened. Also, tools to aid in testing should be used to increase efficiency.

Use of Software Development Tools. As mentioned above, several experts believe the use of software tools may increase efficiency in the test phase of software

development, but many experts believe that using software tools may also increase productivity in other phases of software development.

Rook states, "The earliest tools were concerned with the production of code" (37:7). Today, however, there are many more tools available to the developer that will assist in "specification, design, estimating, planning, documentation and configuration management" (37:7). Bruce and Pederson highlight the importance of software development tools.

With the recent growth in the number of minicomputer - and microprocessor - based systems being developed, this factor has become increasingly important. The cost estimator must consider how the software will be developed, tested, and maintained and what tools will be needed to accomplish these tasks. For systems developed for large-scale computers, a host of compilers, data base managers, editors, display interface packages, flow chart packages, plot packages, utility routines, and test data generation tools are generally available [8:22].

The benefits of using software development tools are numerous and only limited by the number of tools available.

Rook contends that software development tools can

assist in increasing productivity and visibility of work achieved, provide source of data for future proposal preparation, estimation and project planning, and maintain continuity between projects [37:7].

The conclusion is that, when software development tools are available, they should be used to decrease development schedule.

Identification of Resource Requirements. Identifying, as precisely as possible, the resources required to complete

the software development program, many experts believe, is vital to accurately predicting the software development schedule. Brooks relates a story of one Program Manager who found schedules consistently taking twice as long as estimated. After investigation, "the estimation error could be entirely accounted for by the fact that his teams were only realizing 50 percent of the working week as actual programming and debugging time (7:89). This was due primarily to the unavailability of resources because of "machine downtime, higher-priority short, unrelated jobs, meetings, paperwork, company business . . . " (7:89). Resources were not properly identified because "an unrealistic assumption about the number of technical work hours per man-year" was made (7:90).

Pressman also notes that determining the availability of the target machine (machine on which the software will be used) is important when estimating schedule (29:85). Extra time should be taken so that all resources that must be used during the development program can be identified and their availability determined in order that program schedule can more accurately be estimated.

Other Factors. There are other factors that while not elaborated on in detail by experts, they are often mentioned. These factors are listed below.

- 1) Use of a Higher Order Language vs. assembly language (47:159)
- 2) Interface Requirements (15:290)

- 3) Reliability Requirements (8:21)
- 4) Type of software to be developed (47:159)
- 5) Type of contract for the development program (47:159).
- 6) Recurring neglect for software maintenance (38:52).

Conclusion. After a review of the current literature on software development, written by notable experts in the field, twelve factors have been identified as consistently being mentioned by experts as affecting software development schedules. These factors are presented in Table 3 below. Other factors have also been mentioned by experts, but not as frequently, or in as much detail, as these twelve.

Table 3
Factors Consistently Identified By Experts
As Affecting Schedule

1. Lines of code
2. Requirements definition
3. Complexity
4. Work Breakdown Structure
5. Amount of prior planning performed
6. Software development standards
7. Use of management principles
8. Software programmer ability
9. Data base requirements
10. Allowance for test
11. Use of software development tools
12. Identification of resource requirements

Question Two

What is the importance of schedule risk to a software development program?

- a) **To what degree do program managers consider schedule risk when estimating the cost of a software development program?**

The first part of this question was answered by a review of current literature. The sub-part to this question was determined after interviewing program managers and engineers from the various product divisions of AFSC who have had experience in software development and experts in the field of software development in private industry.

Review of Literature. To determine the importance of schedule risk on a software development program, one can examine what program managers and engineers who have managed or who are currently managing software development programs have said about schedule risk and how it has affected their programs. As seen earlier in the review of literature supplied in answer to Investigative Question One, software development program managers/engineers and experts have shown a keen interest in schedule from their readiness to discuss the factors that affect schedule in their writings. Because many of these people are very concerned with the factors that affect schedule in their development programs, it would also seem they would be concerned about the schedule risk (or the probability of the program not being completed on time) associated with their program.

For a software development program manager, the reasons for placing importance on schedule risk become intuitively obvious. The first and far most important reason is allocation of resources. When the schedule of a software development program is set, resources are allocated to the

fulfillment of this schedule. If the schedule slips, more time (an allocated resource) is required and subsequently more resources that were not planned on being allocated to the program must be allocated. These later allocated resources are usually more costly because of the fact they were not allocated for in the first place..

Bruce and Pederson emphasize the importance of schedule risk when they contend one of the major problems in estimating software development costs is the "high level of risk and uncertainty in the estimate" (8:16). They believe that schedule risk and uncertainty are basically attributable to three factors. These factors are: 1) requirements are subject to change; 2) innovation may be required during the development process; 3) risks are inherent in the software development process because errors, which are inevitable, may cause iteration over prior activities (8:16).

Rook has slightly different ideas on the factors that comprise schedule risk. He says the sources of risk can be placed in three main categories (37:8). These categories are: 1) perturbations, which he defines as requirements change, and detection of problems, errors and failures; 2) personnel, defined as the wrong people available, and too many/too few people available; 3) project environment, which comprises an undefined methodology, unknown quality, errors

detected late, and inadequate control, technical skill, support and visibility (37:8).

Donald J. Reifer, author of "The Software Engineering Checklist," also relays the importance of schedule in his writing. In his article, he lists schedule risk as one of the top items management should place on their software engineering checklists (35:127). He says the manager should ask, "Is the software development schedule reasonable and has adequate time been allocated for test?" (35:127). He believes the typical software development schedule does not allocate sufficient time for test and subsequently increases the program's schedule risk (35:127).

Thomas H. Bruggere, author of "Software Engineering: Management, Personnel and Methodology," gives his perspective on the importance of examining schedule risk. He says that schedule risk should be examined throughout every phase of the software development life cycle.

Problems that are discovered during one phase of the project must be fed back to an earlier phase to be fixed. Obviously, the later a problem is discovered and the farther back it must go to be solved, the more expensive the solution [9:25].

He also emphasizes that because schedule risk is an important area for examination, program managers should utilize the review process to ensure design goals and specifications are being met (9:26). He also believes that program managers should closely track schedules to ensure the project will be completed within allowable time limits (9:26).

Driscoll states there are two areas where the program manager can protect the program from cost and schedule impacts of changes and thus reducing schedule risk. These areas are planning and configuration management (19:47). Driscoll explains substantive, early planning can work to reduce schedule risk by "providing schedule flexibility, and adequate computer size" (19:47).

The second reason those involved with software development programs are concerned with schedule risk is political. It becomes a great embarrassment to all those involved when a project slips its schedule. Often the success of a project is not judged by its total quality but by whether or not the project was completed on time and on cost.

Dr. Fred Brooks was part of the management team charged with developing the massive software for the IBM 360 system. He commented after the project's completion, "the effort cannot be called wholly successful . . ." (7:78). Reasons Brooks cited for the project's failure were "the product was late, it took more memory than planned" and the "costs were several times the estimate" (7:78). Brooks references schedule as the first factor contributing to failure in the OS/360 project and demonstrates the importance software development program managers, and those for whom the software is being developed place on schedule.

Howes contends that one way to counteract schedule risk is to develop a proven methodology for conducting the software development and stick with it (23:34). He believes schedule risk can be reduced by using a proven methodology to develop a WBS (23:34). From this WBS cost and schedule estimates can be made. The WBS can also be used to produce a baseline for more accurately measuring the progress of the schedule (23:34).

Discussion of Interview Results. To what degree do program managers and engineers consider schedule and schedule risk? The answer to this question was deduced from an examination of two areas: 1) the degree to which program managers/engineers consider, in their estimates, the factors that were determined to affect schedule risk; 2) the importance program managers/engineers place on the factors they have determined from their experience affect schedule risk. By examining the concerns of those in the field toward the factors that affect schedule, a deduction can be made as to how schedule is considered when developing a software development program estimate. The interviewees were also asked other questions related to software development (see Appendices A & B), and the results of these questions will be briefly discussed also.

Twenty people, comprised of program managers and engineers from the various product divisions of AFSC, and persons considered to be expert or experienced in the area of

software development in private industry were interviewed using the questionnaires in Appendices A & B. Their opinions on the factors affecting schedule and the importance placed on schedule risk were as varied as their backgrounds. The following paragraphs are a summation of the interviews. Note: While conducting these interviews, it was made clear to those being interviewed that it was not the intention of this author to quote persons' opinions specifically; however, the interest here is to get a general consensus on answers to the questions asked in the interview. Particular DOD software development programs are discussed in a general sense, and not specifically identified.

By far, changes in requirements was identified most as having the greatest impact on the schedule of a software development program; however, reasons given by respondents for these changes in requirements varied.

Those interviewed who were contracting with the government to develop software unanimously identified changes in the specification of requirements by the user as being the reason for requirements changes during software development. The contractors often stated that in the early stages of software development, the users do not know exactly what they want the software to perform. In fact, users, at times, may not even finalize the requirements until the later phase of Critical Design Review (CDR).

DOD program managers and engineers also gave reasons for requirements changes. Contrary to what the contractors have stated, one respondent indicated that the problem was the contractor often did not understand what the user wanted. Another stated that requirements changed because there was a change in the standards for the weapon system for which the software was being developed. This appears to be a unique situation. Other reasons for requirements changing were changes in the hardware for which the software was being developed and changes in concerns for the performance of the software (e.g., safety concerns on an avionics system).

One representative of a company that markets a software development program estimation model stated that while he worked for the DOD he was involved in a software development program that actually underran its schedule. The representative said the reason the program did not overrun its schedule was because the schedule was set at no greater than 14 months from the beginning of the development program. The program manager of the program also stated that there would be no changes in the software requirements allowed, in other words, no Engineering Change Proposals (ECPs). This rule was held throughout the development process, and as a result, no delays in schedule occurred as a result of requirement changes. In fact, as was noted earlier no delays occurred at all and the program was completed slightly under schedule.

This example raises an important point. By not allowing requirements to change once a software development program has started, this example has shown that it is possible to meet or underrun an estimated schedule.

The complexity of the software being developed was another factor often mentioned by the respondent as affecting the software development schedule. Complexity in itself is a very broad term. Depending on which software development estimation model is used, complexity can include such areas as the structure of the software (i.e., one module versus a string of modules that must be integrated), what type of language is used (i.e., higher order versus assembly) and what are the display requirements (i.e., simple input/output versus interactive). Generally, the more sophisticated these variables become, the more complex the software. One respondent stressed that complexity is a subjective factor because it is basically one person's opinion. A programmer experienced in the more sophisticated software complexity variables may not rate the software being developed as complex as a programmer with less experience. The respondent went on to suggest that the development of industry standards in the area of determining the complexity of a software program would be beneficial.

The number of development sites, and whether or not the hardware and software are being developed concurrently, are two more factors mentioned by several interviewees as greatly

affecting the schedule of a software development program.

One respondent said that in a typical system program office (SPO) scenario there will be a very general specification for the software and a very general specification for the hardware. Both the hardware and software will be developed at different development sites. Problems that affect schedule occur when the hardware and software are complete and the integration of the two is attempted.

Also mentioned by the respondents as affecting schedule was experience level. The experience of the programmer was mentioned most often. Examples of programmer experience are the degree to which the programmer is familiar with the language being used to develop the software, and the experience the programmer has with similarly structured programs. One respondent also said that the experience the contractor as a whole has with the type of software being developed will greatly affect schedule.

The use of software tools was one more factor mentioned frequently by the interviewees as affecting software development schedule. Many agreed that if good software tools are available, and the programmers are trained in their use, schedule can be reduced considerably.

Surprisingly, the number of lines of code and type of code being developed were factors not mentioned often by the respondents as affecting schedule. This is surprising because of the heavy dependence of estimation models on

predicted lines of code for determining schedule. The response may be an indication that estimators' ability to predict the number of lines of code for a software development program is improving, and, subsequently, this factor has less affect on schedule. One respondent did note that software that can be developed from reusable code and that requires simple inputs/outputs, will help to minimize schedule delays.

Another factor mentioned as affecting schedule was whether or not the contract is a military contract because of the additional documentation and formal reviews required by military contracts. One respondent noted that his firm continuously experienced schedule slippage because of the time it took to get documents reviewed and approved by government personnel.

Finally, the factors time and memory constraints, the facilities available, the database requirements, whether the target computer was also the development computer, and use of management principles were also briefly mentioned by one or more respondents as factors which affect schedule.

One aspect that was noticeable throughout all the interviews was that the interviewees were all very adamant about the factors they believe affect schedule. The respondents also agreed that stronger control of these factors would work to reduce schedule. This gives the indication that software development program

managers/engineers and experts in the software development field place importance on determining the factors that affect development schedule, and effectively controlling these factors.

In addition to being questioned on the factors that affect schedule in software development, the interviewees were asked other questions concerning software development in order to obtain a more rounded picture of what is going on in the field. The results of these additional questions are discussed below.

The interviewees were asked typically, what type of software programs they had experience with. The majority had experience with large software development programs (large being greater than 20000 lines of code). The programs under development ranged from management information systems to avionics systems to flight control systems and simulators.

The interviewees were also asked which software development cost models they were familiar with. All of the respondents were familiar with, or had heard of, the Boehm COCOMO model. Many were also familiar with System-3 and Price-S. Other models mentioned were the Putnam SLIM model, the Tecolote Research, Inc. model and the Ballistic Missile Office (BMO) model.

The respondents were also asked whether they thought a schedule risk factor, which would affect the probability of the schedule being predicted by the model actually occurring,

should be incorporated into the cost model; or should schedule be predicted by combining weights of other factors. The majority of the respondents felt that the probability of meeting the schedule of the software program should be determined from a combination of factors, and not from a single schedule risk factor input by the user.

Finally the respondents were asked to rate the correlation of all of the input factors used in the five models being analyzed to schedule (see Appendix A, Attachment 1 for display of the factors and rating scale). The results from this question were tallied. Requirements volatility was identified by the respondents as the estimation input parameter having the highest correlation with schedule. Requirements volatility is an input parameter for PRICE-S, SoftCost-R, SPQR/20 and System-3 and the updated COCOMO model. The top ten input parameters identified by the interviewees as having the highest correlation with schedule are summarized in Table 4.

Table 4

**Model Input Parameters Identified As Having
The Highest Correlation With Schedule**

1. Requirements Volatility
2. Amount of Hardware Under Concurrent Development
3. Effort Expended During Integration and Testing Phase
4. Development Computer Accessibility
5. Applications Experience With Similar Projects (Tie)
5. Deliverable Lines of Source Code Excluding Documents
6. Schedule Constraints

Table 4 (cont.)

Model Input Parameters Identified As Having
The Highest Correlation With Schedule

7. Complexity of the Logical Design
(Tie)
7. Development Computer Availability
8. Number of Lines of New Source Code
9. Level of Interface With Other Projects or
Organizations
10. Logical Complexity

Question Three

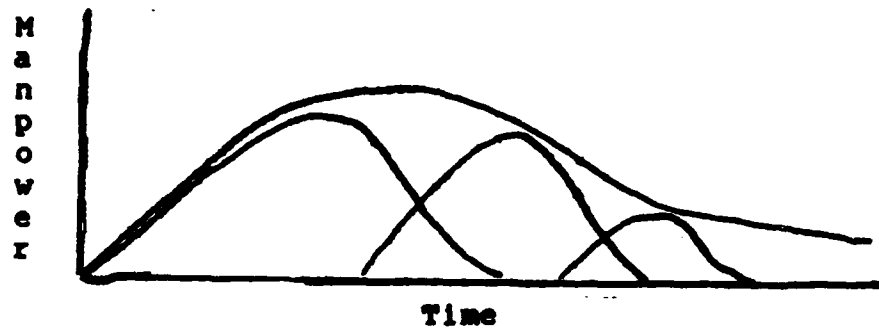
What methods for determining schedule are currently used in software development cost models and have they been tested and/or validated?

- a) Which of these methods appear to be most valid?

Software Development Schedule Theory. Most software estimation models rely on the theory of the Rayleigh-Norden curve as the basis in determining program schedules. Lord Rayleigh, the British Nobel Laureate, originally described the curve that is now used to depict the software project life cycle pattern (32:4). Peter Norden of IBM was the first to relate the software project life cycle to the Rayleigh curve (36:11).

Norden showed that complex research and development projects are composed of overlapping phases of well defined manpower build-up and phase-out (27:80). He calls this relationship the Life-Cycle Manpower Model (27:79). These cycles, when placed together, comprise a larger cycle in its entirety (27:80). This bell-shaped curve is called the Rayleigh-Norden curve (see Figure II).

Figure II
Rayleigh-Norden Curve (27:80)



Typically the Rayleigh-Norden curve has a long tail to the right. Reifer states, "The fact that these cycle curves have long tails explains why projects slip. When the project is 90% done in work, it is only 2/3 done in time" (36:11).

Norden says each cycle can be described by the equation:

$$y' = K e^{-at} \quad (4)$$

where,

- y' = manpower utilized each time period,
- K = total cumulative manpower utilized by the end of the project,
- a = shape parameter (governing time to peak manpower),
- t = elapsed time from start of cycle,
- e = the base of the natural system of logarithms (27:80).

Norden states that because the linking relationships of the cycles have been "encouragingly stable" over a wide range of projects for a number of years, projections of manpower and time requirements can be made on the basis of these

cycles (27:84). This is precisely what many companies have done in developing their estimation models.

Review of Models Selected for Analysis. To answer this investigative question, data was run on five software development estimation models currently in use by the DOD and private industry. The models were selected based on their availability for use with this research effort.

The models selected for discussion and analysis were:

- 1) Constructive Cost Model (COCOMO) (4:4),
- 2) Price-S (34:I-1),
- 3) Softcost-R (42:1-1),
- 4) System-3 (13:1-1),
- 5) SPQR/20 (39:1).

The Putnam SLIM estimation model was not selected for analysis because it was not easily accessible for use.

Description of the Data. To evaluate these models for their accuracy in predicting the schedule of a software development program, data from an actual software development project was run on each of the models. The data is presented in Appendix C. Each model determined what the schedule for the project should be and this result was compared to the actual schedule for the project.

The data on the software development project was obtained from the Software Cost Data Base which was compiled by Paul G. Funch of the MITRE corporation for use at MITRE

and the Electronic Systems Division, Hanscom AFB, MA and is not authorized for public release (22:v).

The Software Cost Data Base was compiled in 1987 for two reasons. First, analysts at MITRE and ESD have found that software estimation models are often based upon data bases that may not represent ESD programs, "which are typically large, embedded, complex, highly reliable, real-time, military applications" (22:v). Second, "the accuracies of cost estimates are not evaluated at the completion of projects since there is not budgetary justification to do so" (22:v). Therefore, little historical data exists to "enhance the personal experiences of cost analysts and software project managers" (22:v).

The Software Cost Data Base consists of 26 projects and comprises "a total of 110 computer software configuration items (CSCI)" (22:v1). The sizes of the projects widely vary. The largest project in the data set has well over 1 million lines of code and the smallest has 9000 lines of code (22:A-11).

One average project was selected from this data base to run as a test case in each of the models. Number of lines of code (LOC) was the variable used as the basis for determining an average project. Projects at the high and low extreme ends for LOC were eliminated from the calculation and LOC for the remaining projects were averaged together. The average number of LOC for the data base was 178,663. The project

having an LOC count closest to this number was project #24 with 185,600 LOC; therefore, project #24 with modules A, B, C, and D was chosen as the test project to be run on each of the models.

Analysis of Results by Model. The data from project #24 was used as input to each of the software development estimation models selected for analysis. The following paragraphs are a discussion of the resulting output by model.

COCOMO. COCOMO, as noted earlier in Chapter II, is a non-proprietary software development model developed by Barry Boehm. After accumulating the data for the data base, Paul G. Funch, developer of the Software Data Base, used the data to evaluate COCOMO's ability to estimate the type of software programs commonly developed at ESD.

Funch examined five of the COCOMO equations. Among the equations Funch examined were the effort and schedule equations for the embedded mode of the Basic COCOMO Intermediate COCOMO models (22:vii). These equations were selected for use in this research. Recall, the Basic COCOMO equation for effort in the embedded mode is:

$$MM = 3.6(KDSI)^{1.20} \quad (5)$$

where MM is the number of man-months required to develop the software product and KDSI is the number of thousands of delivered source instructions (4:75). The Intermediate COCOMO Nominal effort estimating equation for the embedded mode is:

$$MM_{nom} = 2.8(KDSI)^{1.20} \quad (3)$$

where MM_{nom} is the nominal estimate of man-months required to develop the software product (4:117). With the Intermediate COCOMO model, this nominal estimate is adjusted "by applying effort multipliers determined from the project's ratings with respect to the other 15 cost driver attributes" (4:117).

The Intermediate COCOMO adjusted estimating equation is:

$$MM_{adj} = (MM_{nom})(EAF) \quad (6)$$

where EAF are the product of the effort adjustment factors found by rating the 15 cost driver attributes (4:120). To determine schedule or duration (TDEV), the result of this equation is input back into the Basic equation for schedule:

$$TDEV = 2.5(MM)^{0.32} \quad (4:75). \quad (7)$$

Using the Software Cost Data Base, Funch recalibrated the Basic and Intermediate COCOMO estimating equation by "fixing the exponent to the value established by Boehm and calculating the best fit coefficient" (22:vi). The recalibrated Basic COCOMO equations for effort and schedule are:

$$MM = 6.5(KDSI)^{1.20} \quad (8)$$

$$TDEV = 3.8(MM)^{0.32} \quad (9)$$

The Intermediate COCOMO estimating equations as recalibrated by Funch are:

$$MM_{nom} = 3.3(KDSI)^{1.20} \quad (10)$$

$$MM_{adj} = 3.3(KDSI)^{1.20} (EAF) \quad (22:ix). \quad (11)$$

Funch states that the Boehm schedule equation for the embedded mode usually underestimated durations of projects in the ESD/MITRE data base (22:vii). Funch recalibrated the coefficient using 12 projects (22:vii).

Funch states that when this recalibrated schedule equation (shown above) was used with the data in the Software Cost Data Base, "the estimates were found to be within 30% of the actuals 67% of the time" (22:vii). Funch notes that this performance is nearly identical to that of the Boehm schedule equation on the COCOMO data base (22:vii). Also, Funch states, "The Boehm COCOMO schedule equation for the embedded mode underestimated the durations of all but one project in the ESD/MITRE Data base" (22:viii). He says, however, "the recalibrated equation . . . predicts schedules 52% longer than the Boehm equation" (22:viii).

Similarly, the Boehm Basic model effort equation for the embedded mode, Funch found, tended to underestimate actual subsystem efforts (22:vii). The coefficient for this equation was recalibrated using 17 subsystems (22:vii). Funch notes that the value of 6.5 differs significantly from the Boehm coefficient; however, when the recalibrated coefficient effort equation was used, "the estimates were found to be within 20% of the actuals 35% of the time" (22:vii). Funch states this performance "exceeds the performance of the Boehm Basic equation on the COCOMO data base, which had actuals within 20% of the estimates only 21%

of the time" (22:vii). Funch states that the recalibrated Nominal Intermediate model equations for effort in the embedded mode (and semidetached mode) provided "marginally better fits to the subsystem data" (22:vii).

Because of the established ability of the recalibrated equations to better estimate ESD-type software development projects, the data from project #24 was run on the recalibrated equations. The recalibrated Nominal Intermediate model equations,

$$MM_{nom} = 3.3(KDSI)^{1.20} \quad (10)$$

$$MM_{adj} = 3.3(KDSI)^{1.20} (EAF) \quad (11)$$

were used to calculate effort and the recalibrated Basic COCOMO equation,

$$TDEV = 3.8(MM)^{0.22} \quad (9)$$

was used to calculate schedule or duration for project #24.

Appendix D gives the ratings and effort multipliers given to each of the COCOMO Intermediate model cost driver attributes. The effort multipliers were used to compute the Effort Adjustment Factor. The following is the calculation for schedule using the Funch recalibrated COCOMO Intermediate equation for effort and the recalibrated Basic COCOMO equation for schedule:

$$EAF \text{ for Project \#24} = 2.83 \text{ (see Appendix D),}$$

$$KDSI \text{ for Project \#24} = 30200,$$

$$MM_{nom} = 3.3(30.2)^{1.20} = 250.16$$

$$MM_{adj} = (250.16)(2.83) = 707.96$$

$$TDEV = 3.8(707.96)^{0.22} = 12.5 \text{ months}$$

Actual duration for Project #24 was 26 months, twice that of the COCOMO model prediction.

PRICE-S. The project data was run in the PRICE-S MODE 2--CSCI with components model because the data for the project was available by components A, B, C, and D. This author felt the PRICE-S model was not as user-friendly as the other commercial models used in this analysis; the data is more difficult to input interactively because the model does not prompt the user for the inputs. The user can also input data from a separate file that must be written. The data file is not difficult to write, but it requires more time than just inputting the values directly into the model and having the model immediately compute the result.

The values input from the data for each variable in the PRICE-S model are given in Appendix E. The output from the model is given in Appendix F. Recall, the actual duration of Project #24 was 26 months. This is number accounts for the time frame from System Design Review (SDR) through Formal Qualification Test (FQT). The PRICE-S model estimates the duration from System Concept (two phases before SDR) through Operational Test and Evaluation (OTE) (three phases after FQT). For the corresponding time frame for the project data (SDR through FQT), PRICE-S estimated the duration to be 30.4 months.

SoftCost-R. SoftCost-R, as described earlier, is a software development program estimation model developed and marketed by Reifer Consultants, Inc. It is a menu driven model that this author found to be very user friendly primarily because inputs are done interactively. To initialize an estimate, SoftCost-R asks the user to answer questions in four different menus: Management Factors Menu, Staffing Factors Menu, Complexity Factors Menu, and Environmental Factors Menu (42:11).

The inputs by variable used for Project #24 with SoftCost-R are given in Appendix G. The output given for the data inputs is presented in Appendix H.

SoftCost-R predicted the duration of the project to be 21.1 months. Recall again, the actual duration was 26 months.

SPQR/20. SPQR/20 asks the user to input values for variables interactively through three menus. The first menu is titled SPQR/20 Input Variables. It contains general questions about the project such as what type of estimate is required (i.e. cost versus schedule or both), what is the maximum staff size and what is the average work week (39:10).

The second menu is titled Environmental Inputs. It contains questions about area such as program design, staff experience and amount of reusable code (39:18).

The third menu is titled Complexity and Source Code Input Parameters and contains questions about the complexity of the new code being developed (39:21).

There are two additional menus. The first, Base Code Input Parameters is used when the software being developed is an enhancement or maintenance to existing software (39:26). The second is titled Function Point Inputs and is used when the user wants SPQR/20 to predict the number of new lines of code that will be developed (39:27).

The SPQR/20 model will estimate development schedule from the planning activity through the integration/test activity (39:36). The duration estimated for the requirements, design, coding and integration/test activities was used in this analysis because it most closely describes the time frame given for the duration of Project #24. The data which was input into the SPQR/20 model is given by variable in Appendix I. SPQR/20 estimated the duration of the project to be 29.21 months. Recall again, the actual duration for Project #24 was 26 months. The summary output from the model is shown in Appendix J.

System-3. System-3 is also a fairly user friendly model that is also menu driven. One of its menu features allows the user to break down a large project being estimated by groups within the project, by tasks within the group and by elements within the tasks. For this analysis, the data was run at the project level because the specific information

required to run modules A, B, C, and D at the group level was not available.

System-3 also has eight menu screens for input of variable data. These screens are titled Developer Technology, Environmental-Computer, Environmental-Product, Environmental-Support, Size & Complexity Summary, Development Constraints, Reuse-Rebuild Impact, and Financial Factors (13:20). The values input at each of these menu prompts is given in Appendix K.

System-3 calculated a minimum time estimate for development time and integration time as 24.69 months. System-3 also calculated requirements time, however, this time was not included because it was not included in the actual duration time of 26 months for Project #24. The output given for the Project #24 data is given in Appendix L.

Summary of Results. The schedule times given by the five models analyzed for the Project #24 data is summarized in Table 5 below. System-3 most closely estimated the duration for Project #24 with a difference of 1.31 months below the actual duration. The model that predicted farthest from the actual was the ESD recalibrated COCOMO model with a difference of 13.5 months under the actual schedule. All of the other four models estimated duration within five months of the actual schedule.

TABLE 5

Estimated Project Duration by Model (in months)

	<u>Months</u>	<u>Actual</u>	<u>Difference</u>
ESD Recalibrated COCOMO	12.50	26	13.50 (under)
PRICE-S (MODE 2)	30.40	26	4.40 (over)
SoftCost-R	21.10	26	4.90 (under)
SPQR/20	29.21	26	3.21 (over)
System-3	24.69	26	1.31 (under)

Question Four

In cost models, what is the significance of schedule risk?

- a) Is schedule risk an independent variable or does its significance depend on the value of other independent variables in the cost models such as size of the program, number of programmers required, or level of software sophistication required?

The models selected for this analysis were the same ones used in the analysis for investigative question three. As noted earlier, all of the models, except the COCOMO model were found to contain proprietary algorithms. The main difference between each of the models in their determination of schedule was the factors the models used as inputs to determine a program's estimate.

The Boehm COCOMO model, as mentioned earlier, has 15 effort multipliers. All the multipliers influence schedule to some degree because they are all used to determine effort which, in turn, is used in the COCOMO schedule equation. By

far, the most significant influence on schedule, however, is lines of code. The predicted number of lines of code is used directly to determine effort.

After examining the range of values Boehm assigns each of the effort multipliers (see Appendix M for values), it is evident that those with the possibility of most significantly influencing effort (and schedule) in descending order of influence are: execution time constraint, product complexity, main storage constraint, analyst capability, programmer capability and required software reliability. The other nine effort multipliers have a relatively lower influence on effort.

PRICE-S uses a series of equations to determine the schedule of a software development program. PRICE-S computes schedules for design (preliminary and detail), code (unit and CSC), and test (CSCI). Most of actual values used by these equations are proprietary, but the factors used to compute the equations are not. The schedule equations for PRICE-S rely heavily on these factors:

- 1) NEWD - represents amount of code requiring new design (28).
- 2) NEWC - represents amount of new code required (28).
- 3) SLOC - source lines of code required (34:11-A10).
- 4) APPL - the user defined application value that describes the application mix of the software (34:11-A12).

- 5) UTIL - the factor describing the fraction of available hardware cycle time or total memory capacity used (34:11-A4).
- 6) CPLX1 - the factor which provides a quantitative description of the relative effort of complicating factors on the software development task. Complicating factors are product familiarity, personnel skills, software tools, and any unusual factors present in the development environment that affect the development schedule (34:11-A8).
- 7) PROFAC - an empirically derived parameter which includes such items as skill levels, experience, productivity, and efficiency (34:11-A10).
- 8) PLTFM - a value ranging from 0.6 to 2.5 which is a measure of the customer's requirements for portability, reliability, structuring, testing and documentation required for acceptable contract performance (34:11-A2).
- 9) ZTECH - a technology improvement factor (28).

The equation for schedule in the PRICE-S model is

$$\text{Schedule} = \alpha * F * WT^{\beta} * CPLX1^{\gamma} \quad (12).$$

where,

α, β, γ = proprietary values

and,

$$F = K * PROFAC^{\alpha} * (PLTFM/K)^{\beta} * ZTECH^{\gamma}$$

$$WT^{\beta} = SLOCm^{\beta} * S^{\beta} * UTIL^{\beta}$$

where,

K = proprietary value

SLOCm = SLOC * APPL

S = required effort (28).

While PRICE-S uses all these values mentioned in the schedule equation, it appears that the key value that is input first, and the basis for the rest of the equations, is SLOC.

It is not clear what types of equations the remaining three models analyzed; SoftCost-R, SPQR/20, and System-3; use to derive schedule because no indication is given in their documentation. However, after running the models, it is clearly evident to this author that these models also rely heavily on the predicted number of lines of code to predict schedule. One model, SPQR/20 will predict lines of code for the user with a technique discussed earlier called Function Points.

The other inputs to these models (that also affect schedule to some degree that is proprietary) are basically a variation of the 15 effort multipliers developed by Boehm for the COCOMO model. One might assume that the factors known to way heavily on schedule for the COCOMO equation, such as product complexity, analyst capability, and required software reliability, would also weigh heavily in these proprietary models.

In conclusion, it is evident from examination of the equations used in COCOMO and PRICE-S that schedule and schedule risk are significant determinants in calculating these model's estimates. Schedule risk is a function of the values of many variables as can be seen by the variety of

variables used as inputs into all five of these models. For some models schedule is taken into direct consideration when the models ask if there are any schedule constraints involved; however, none of the models analyzed here asks for a single schedule risk input. All appear to derive schedule risk from a combination of the other variables. The heaviest weighted of these variables in determining schedule is number of lines of code.

Question Five

Can any of these methods be combined or incorporated into a new and more accurate method for accounting for schedule risk in cost models?

To address this question, a clarification of the term methods must be made. Methods in this question means the processes through which the five models previously analyzed predict schedule. As noted earlier, this process in all the models except COCOMO is proprietary. Therefore, it is really uncertain to someone not having knowledge of all the proprietary algorithms whether or not any of the methods can be combined or incorporated into a new more accurate model.

However, one can observe from using the models on a test case, that they all basically require the same inputs in various forms. Therefore, because these inputs are the crux of the equations used to predict schedule, more emphasis should be placed instead on more accurately determining the values for these variables. That will happen when (and if)

standards are developed for making the estimation of these variables as objective as possible.

For instance, complexity of the program is a relatively subjective input variable that can vary depending on the experience of the programmer making the estimation and not actually performing the work. The programmer working on the program may have a different value for the complexity of the program. In the same light, methods for accurately estimating another crucial input variable, lines of code, should be investigated as well.

In conclusion, it is uncertain because of the proprietary nature of these equations whether they can be combined or incorporated to make a more accurate model; however, it is certain that methods for more accurately estimating the variables input into these models should be developed.

V. Conclusions and Recommendations

Conclusions

The schedule of a software development program can be affected by a large array of varying factors. This research, however, has identified twelve of these factors as having the greatest impact on the schedule of a software development program. Determination of these factors was made through extensive reviews of literature written by experts in the software development field. The twelve factors identified were: lines of code, requirements definition, complexity, work breakdown structure, amount of prior planning performed, software development standards, use of management principles, allowance for test, use of software development tools and identification of resource requirements.

These factors were confirmed through interviews with DOD program managers/engineers and commercial software development estimation experts as heavily influencing the schedule of software development programs. The factor most often mentioned in these interviews as causing delays in schedule was requirements definition. The interviewees agreed that the requirements of the software development program are either inadequately specified at the beginning of development or they are not firm requirements and subsequently changes causing delays are made during development.

Because changes in requirements definition was identified most often as being the primary cause for software development schedule slippage, software development program managers should investigate ways to more accurately determine the requirements of a program.

Five software development models were analyzed using a data base developed by Electronic System Division of Air Force Systems Command. The five models chosen were an ESD recalibrated COCOMO model, PRICE-S, SoftCost-R, SPQR/20 and System-3. Using data from one ESD software development project as a case study, System-3 was shown to predict its schedule with the most accuracy by predicting the schedule within 1.31 months of the actual. The next most accurate for this case was SPQR/20, then PRICE-S, and then SoftCost-R. The worst predictor of the case's schedule was the ESD recalibrated COCOMO model. Its prediction was 13.5 months under the actual. In summary, it is evident from this case that all of the models analyzed except the ESD recalibrated COCOMO model did very well at predicting the project's actual schedule. These four models' predictions were all within five months of the actual schedule.

An examination of the equations used in the COCOMO model and the PRICE-S model has revealed that these models use a combination of variables to predict schedule. Some are more heavily weighted than others. It can be assumed that is also

the case with other three models analyzed whose equations used to determine schedule are proprietary.

To develop a more accurate model for predicting the schedule of a software development program, emphasis should be placed not on developing new models but on developing better methods for accurately predicting the variables which are input into these models.

Recommendations

Further research in the area of schedule determination for software development programs is needed in order to help estimators more accurately estimate these schedules. The following paragraphs are suggestions for areas of continued research.

Twelve factors were identified as heavily affecting the development schedules of software. The number one cause of delays in schedule identified appears to be inadequate definition of requirements. Further research should be done to determine what are the reasons for changes in requirements definition and how can changes in requirements definition be reduced.

Because of time limitations, only one project, Project #24 data was input into each of the five software estimation models being analyzed and an estimate was calculated. It is realized that one data point may not give a complete picture of the accuracy of each of the models at predicting schedule. Therefore, the remaining twenty-five data points of the ESD

Software Cost Data Base, developed by Paul G. Funch, should also be input into each of the models and allow the models to calculate estimates for these points as well. From this data, the standard deviation of estimates from actual schedules should be calculated for each model. By using twenty-six data points, a more accurate picture of each model's schedule estimation capabilities can be presented.

There are several other software development estimation models available today besides the five models analyzed here. These models should be identified and the data from the ESD Software Cost Data Base should be used as a test case to determine the accuracy of these models in predicting schedule also.

Using the ESD Software Cost Data Base for data, the twelve factors identified in this research as heavily affecting schedule should be regressed against schedule to determine which factors are the most highly correlated with schedule. Once this is determined, a model can be developed from these factors that will predict the schedule for the types of software programs developed at ESD.

APPENDIX A:

INTERVIEW QUESTIONNAIRE
FOR DOD PROGRAM MANAGERS/ENGINEERS

- 1) What type or types of software development programs have you managed recently (i.e. large or small content, what weapon system was the software being developed for)?
- 2) Upon completion, did the software development program overrun or underrun its cost estimate?
 - a) What was the initial estimate for the program?
 - b) What was the final cost of the program?
 - c) Were there changes in the software requirements during the program?
 - 1) If so, at what phase and why?
- 3) On completion, was the software development program on schedule, over schedule or under schedule?
 - a) What was the original schedule?
 - b) What was the final schedule?
 - c) Were there changes in the software requirements?
 - 1) If so, at what phase and why?
- 4) If the software development program ran over or under schedule, in your opinion, what were the factors that caused this over or underrun (i.e. complexity of the software being developed, experience of the programmers, number of programmers used, changes in requirements)?
- 5) Was a software development cost model (or models) used to estimate the cost of the software development program? If so, what model was used?

- 6) Most software development cost models use proprietary algorithms to develop program estimates. In your opinion, how do you believe schedule risk was incorporated into the software development cost model you used to determine the cost of the program (if a software development cost model was used)? In other words, which factors that you input to the model do you feel contributed most to determining the schedule of the program? (Factors for each cost model being analyzed provided in Attachment 1).

NOTE: Factors are given at the end of this Appendix.

- 7) From your experience with software development programs, what factors do you consider important or significant when determining the schedule and/or schedule risk of a software development program?
- 8) Do you think a schedule risk factor, which will affect the probability of the schedule being predicted by the model actually occurring, should be incorporated into the cost model; or do you think schedule should be predicted by combining weights of other factors such as program complexity?

ATTACHMENT 1 (page 1 of 5)

The various input factors listed below are used by the software development program estimating models being analyzed. You are asked to rate each of the factors in terms of to what extent you believe they affect the schedule of a software development program using the following scale:

0	1	2	3	4	5
	Very				Very
Don't	Low	Low	Somewhat	High	High
Know	Correlation	Correl.	Correlated	Correl.	Correl.

Rating Factors

		COCOMO PRICE-S SOFTCOST-R SPQR/20 SYSTEM-3				
	Factors Common to 3 or more models					
	1. Specification (SPEC) = functional, procedural, both or other.		X		X	X
	2. Requirements Volatility (RVOL)		X	X	X	X
	3. Main Storage Constraint (STOR)	X	X	X		X
	4. Computer Turnaround Time (TURN)	X			X	X
	5. Use of Software Tools (TOOL)	X	X	X		X
	6. Applications Experience with similar projects (AEXP)	X	X	X		X
	7. Average Experience with Techniques used (TECH)		X	X	X	
	8. Programming Language Experience (LEXP)	X	X	X		X
	9. Analyst Capability (ACAP)	X	X	X		X
	10. Software Product Complexity (CPLX)	X	X	X	X	X
	11. Deliverable Lines of Source Code Excluding Documents (DSLOC)		X	X		X
	12. Lines of Source Code Documentation (DocLOC)		X		X	X
	13. Reusable Code From Similar Projects		X	X	X	X

ATTACHMENT 1 (page 2 of 5)

Rating	Factor	COCOMO	PRICE-S	SOFTCOST-R	SPQR/20	SYSTEM-3
	Factors Common to 2 models					
	1. Use of Modern Programming Practices (MODP)	X				X
	2. Virtual Machine Volatility (VIRT)	X				X
	3. Execution Time Constraint (TIME)	X		X		
	4. Number of Development Sites (SITES)			X		X
	5. Virtual Machine Experience (VEXP)	X				X
	6. Programmer Capability (PCAP)	X				X
	7. Special Display Requirements (Display)				X	X
	8. Effort expended during the integration and testing phase (EFTit)		X			X
	9. Application Complexity		X			X
	10. Number of lines of new source code		X	X		X
	11. Number of lines deleted from existing modules			X		X
	12. Number of existing modules requiring modification			X		X
	13. Existing LOC = pre-existing code prior to task development			X		X
	14. Monthly Pay Average				X	X
	Unique Factors by Model System-3					
	1. Annual Inflation (Financial)					X
	2. Max Effort-Person Months					X
	3. (Estimate) Probability Required					X
	4. Real Time Operation (RTIM)					X
	5. Rehosting (HOST)					X
	6. Relaxed Schedule (Min. Effort)					X
	7. Requirements - % Development					X
	8. Requirements Effort Complete at Contract Award					X
	9. Resource Dedication (RDED)					X

ATTACHMENT 1 (page 3 of 5)

Rating	Factor	COCOMO	PRICE-S	SOFTCOST-R	SPQR/20	SYSTEM-3
	Unique Factors by Model (continued)					
	System-3 (continued)					
	10. Resource Location (RLOC)					X
	11. Staff - Requirements staff, Project staff, Development staff					X
	12. System (Virtual Machine) Complexity (SYST)					X
	13. Effective Productivity = average lines of code completed by project person per month		X			X
	SPQR/20					
	1. Estimate Scope (Prototype, Module, System)				X	
	2. Project Estimating Goals				X	
	3. Output Metric (man-months, years, etc.)				X	
	4. Exempt Personnel				X	
	5. Average Work Week				X	
	6. Average Work Year				X	
	7. Other Costs (holds costs not estimated by SPQR/20)				X	
	8. Project Class (Personal program through military contract)				X	
	9. Office Facilities				X	
	10. Reusable Code Language				X	
	11. Reusable Code Language Level				X	
	12. Logical complexity				X	
	13. New Code Structure				X	
	14. New Code Data Complexity				X	
	15. New Code Language				X	
	16. Language Level				X	
	17. Base Logical complexity				X	
	18. Base Code Complexity				X	
	19. Database complexity				X	
	20. Base Code Language				X	
	21. Base Code Language level				X	
	22. Base Code Size				X	

ATTACHMENT 1 (page 4 of 5)

Rating	Factor					
		COCOMO	PRICE-S	SOFTCOST-R	SPQR/20	SYSTEM-3
	Unique Factors by Model (continued) Softcost-R					
	1. Number of lines changed in other ways in existing modules			X		
	2. Number of lines deleted as entire modules			X		
	3. Phase of the life cycle software work begins			X		
	4. Expected user involvement in requirements definition			X		
	5. Customer/implementor organizational interface complexity			X		
	6. Level of interface with other projects or organizations			X		
	7. Efficiency of implementing organization			X		
	8. % of programmers doing design and will work development			X		
	9. Type of technical reviews held			X		
	10. Number of I/O items generated per 1000 lines			X		
	11. Overall complexity of the data base architecture			X		
	12. Complexity of the logical design			X		
	13. % of the program in assembly language			X		
	14. % of total task will be easy			X		
	15. % of total task will be hard			X		
	16. Classified Security environment for computer (Y/N?)			X		
	17. Amount of hardware under concurrent development			X		
	18. Development computer accessibility			X		
	19. Development computer availability			X		
	20. Maturity of system and support software			X		

ATTACHMENT 1 (page 5 of 5)

Rating	Factor	COCOMO	PRICE-S	SOFTCOST-R	SPQR/20	SYSTEM-3
	Unique Factors by Model (continued)					
	SoftCost-R (continued)					
	21. Overall adverse constraints on program design			X		
	22. % of program which is real-time and multi-tasking			X		
	23. Software adapted to multiple environments (Y/N?)			X		
	24. Adaptation required to change from development to operational environment (Y/N?)			X		
	PRICE-S					
	1. INTEG1 = the level of difficulty of integration and testing the CSCIs to the system level		X			
	2. SCHEDULE = start of the system software development activities and completion dates for activities which support the software development phases		X			
	3. LANG = source language to used		X			
	4. CPLX2 = quantitative description of the relative effect of complicating factors on the software development task caused by hardware/software interactions. Factors include new hardware development and hardware developed in parallel		X			
	COCOMO					
	1. Required Software Reliability (RELY)	X				
	2. Data Base Size (DATA)	X				
	3. Schedule Constraints (SCHED)	X				

APPENDIX B:

**INTERVIEW QUESTIONNAIRE
FOR SOFTWARE DEVELOPMENT EXPERTS
(DOD AND COMMERCIAL)**

- 1) Which software development cost model are you familiar with? (Choose one or more)
 - a) COCOMO
 - b) Price-S
 - c) Putnam-SLIM
 - d) Softcost-R
 - e) System-3
 - f) SPQR/20
- 2) Upon completion, did the software development program you have been involved with, and used this model(s) for an estimate, overrun or underrun its cost estimate?
 - a) What was the initial estimate for the program?
 - b) What was the final cost for the program?
 - c) Were there changes in the software requirements during the program?
 - i) If so, at what phase and why?
- 3) Same as Questionnaire for DOD Program Managers/Engineers.
- 4) Same as Questionnaire for DOD Program Managers/Engineers.
- 5) Same as Questionnaire for DOD Program Managers/Engineers.
- 6) In your opinion, how is schedule risk incorporated into this software development cost model? In other words, which factors input in the model contribute more to determining the schedule of the program? (Factors for each cost model being analyzed are provided in Attachment 1).

NOTE: See Appendix A, ATTACHMENT 1 for questionnaire attachment.
- 7) Same as Questionnaire for DOD Program Managers/Engineers.
- 8) Same as Questionnaire for DOD Program Managers/Engineers.

APPENDIX C:

Mitre Project Data

Project # 24 (A,B,C,D)	24	A	B	C	D
<u>Description of Factors</u>					
Descr = Mission Description					
1= C3 system	*	2	2	2	2
2= radar system					
3= simulation					
4= training system					
SWfnc1 = Major Software Function	*	3.8	3.8	11.3	4.9
SWfnc2 = Major Software Function		4.1	15.3	13.3	5.3
SWfnc3 = Major Software Function		5.3	17.2	17.1	7.1
#CSCI = Number of CSCIs	*	6	6	6	6
Simul = CSCIs simultaneously resident on Target Computer are assigned same number.	*	*	*	*	*
Specif = Specification					
1= functional	*	1	1	1	1
2= procedural					
3= 1 + 2					
4= Other - match analogous programs					
Design = Design Method					
1= Top down	*	1	1	1	1
2= bottom up					
3= iterative enhancement					
4= hardest first					
5= 1 + 2					
6= 1 + 3					
7= 2 + 3					
Develop = Development Method					
Same as above	*	1	1	1	1
Coding = Coding Method					
1= structured code	*	1	1	1	1
2= other, pseudostructured code					
3= other, by modules					

Mitre Project Data (page 2 of 18)

Project # 24

24 A B C D

Description of Factors

Testing = Testing Method

1= top down (stubs)

2= bottom up (drivers)

3= specification driven

4= structure driven

5= Other, input/output

6= other, module level testing and applications software testing

7= 1 + 2

8= 1 + 3

9= 2 + 3 + 4

10= 3 + 4 + 6

11= 1 + 2 + 3 + 4

8 8 8 8 8

V-V = Validation/Verification Method

1= peer review

2= walk throughs

3= proof

4= none

5= 1 + 2

6= 1 + 2 + 3

* 5 5 5 5

Formal = Formalisms Methods

1= program design language

2= HIPO charts

3= flowcharts

4= 1 + 2

5= 1 + 3

6= 2 + 3

7= 1 + 2 + 3

* 1 1 1 1

Software Development Tools Used

Tvlow = # of tools used from list:

Assembler

Basic Linker

Basic Monitor

Batch Debug Aids

* 0 0 0 0

Mitre Project Data (page 3 of 18)

Project #24 (A,B,C,D) 24 A B C D

Description of Factors

Tlow = # of tools used from list: *	2	2	2	2
Higher Order Language Compiler				
Macro Assembler				
Simple overlay linker				
Basic Source editor				
Language Independent				
Monitor				
Basic Library Aids				
Basic Database Aids				
 Tnom = # of tools used from list: *	 2	 2	 2	 2
Real-time or Time-sharing				
Operating system				
Extended Overlay Linker				
Database Management System				
Interactive Debug Aids				
Simple Programming				
Support Library				
Interactive Source				
Editor				
 Thigh = # of tools used from list:*	 1	 1	 1	 1
Virtual Memory				
Operating System				
Database Design Aid				
Simple Program Design				
Language				
Performance Measurement				
and Analysis Tools				
Programming Support				
Library with Basic Con-				
figuration Management Aids				
Set-use Static Analyzer				
Basic Text Editor & Manager				
Program Flow and Test Case				
Analyzer				
File Manager				

Mitre Project Data (page 4 of 18)

Project #24 (A,B,C,D)	24	A	B	C	D
<u>Description of Factors</u>					
Tvhigh = # of tools used from list:	*	3	3	3	3
Full Programming Support Library					
Documentation System					
Project Control System					
Requirement Specification Language and Analyzer					
Extended Design Tools					
Automated Verification System					
Fault Report System					
Crosscompilers					
Instruction Set Simulators					
Display Formatters					
Data Entry Control Tools					
Communications Processing Tools					
Conversion Aids					
Structured Language Tool					
Tother = # of other Tvhigh tools used not listed above.	*	0	0	0	0
Tcalc = Calculated Tool Parameter (Ranges from 1 = vlow to 5 = vhigh)	*	3.26	3.26	3.26	3.26
MODP = Use of Modern Programming Practices	*	.91	.91	.91	.91
1.24 = vlow					
1.1 = low					
1 = nom					
.91 = high					
.82 = vhigh					
Begin = COCOMO Project starting date (SDR) mm.yy	9.82	9.82	9.82	9.82	9.82
End = COCOMO Project completion date (FQT)	10.84	10.84	10.84	9.84	10.84
Duration = Project duration	26	26	26	25	26

Mitre Project Data (page 5 of 18)

Project #24 (A,B,C,D) 24 A B C D

Description of Factors

Actual Schedule Milestones

Note: PDR = Preliminary Design Review
 SDR = System Design Review
 CDR = Critical Design Review
 Int = Beginning of Integration
 and Testing
 FQT = Formal Qualification Test # of months

PD = PDR - SDR (or Contract Award) *	7	7	7	7
DD = CDR - PDR *	6	6	6	6
CUT = Int-CDR *	6	6	6	6
IT = FQT (or Complete CPCI integration) - Int *	6.5	6.5	5.5	6.5
TOT = Total Project Time (PD + DD + CUT + IT) *	25.5	25.5	24.5	25.5

System Documentation
(Project Level)

DOC1 = System Engineering Management Plan (# pages) *	*	*	*	*
DOC2 = Computer Program Development Plan (# pages)	33(total)			
DOC3 = System Test Plan (# pages) *	*	*	*	*
DOC4 = Other Documentation (# pages) *	*	*	*	*
SCnum = Software Change history: # of requirements changes during software development phases = # of changes approved. *	*	*	*	*
SCloc = Change in DSLOC resulting from the requirements changes *	*	*	*	*
SCsm = Change in effort resulting from the requirements changes = # staff-months *	*	*	*	*

Mitre Project Data (page 6 of 18)

Project #24 (A,B,C,D) 24 A B C D

Description of Factors

RVOL = Requirements Volatility Not given
 .91 = low (Essentially none)
 1.0 = nom (small, noncritical
 redirections)
 1.19 = high (Occasional moderate
 redirection)
 1.38 = vhigh (Frequent moderate
 or occasional major)
 1.62 = ehigh (Frequent major
 redirection)

Development and Target Computer Data

VIRT = Virtual Machine Volatility
 .87 = vlow (no major changes;
 minor change every 12
 months)
 .87 = low (maj. changes every .87 .87 .87 .87 .87
 12 mos; minor changes
 every month)
 1.0 = nom (maj. changes
 every 6 mos minor
 changes every 2 weeks)
 1.15 = high (maj changes every
 2 mos; minor changes
 every week)
 1.3 = vhigh (maj changes every
 2 weeks; minor changes
 every 2 days)

STOR = Main Storage Constraint
 (CPU Memory Percent Utilization)
 Maximum % of processing time used
 by any group of CSCIs executing
 concurrently on single machine.
 1.0 = nom (<=50%)
 1.06 = high (51-70%)
 1.21 = vhigh (71-85%)
 1.56 = extra hi (86-95%) 1.56 1.56 1.56 1.56 1.56

Mitre Project Data (page 7 of 18)

Project #24 (A,B,C,D)

24 A B C D

Description of Factors

TIME = Execution Time Constraint
(Execution Time Percent
Utilization)
Maximum % of processing
time used by any group of
CSCIs executing concurrently
on single machine

1.00 = nom (<=50%)

1.11 = high (51-70%)

1.30 = vhigh (71-85%)

1.66 = ehigh (86-95%)

1.3 1.3 1.3 1.3 1.3

MemCE = CPU Memory Constraint
Evaluation

Measures required to
satisfy the reserve
memory requirement

1 = no mem economy measures
req'd

2 = some overlay use or
segmentation req'd

3 = extensive overlay
and/or segmentation
req'd

3 3 3 3 3

4 = complex memory
management economy
measures req'd

TimCE = CPU Time Constraints Evaluation

Percentage of software
that requires special
coding effort to enhance
timing performance

1 = no software is cpu
time constrained

2 = <25% of source code
is time constrained

3 = 25-50% of source code
is time constrained

4 = 50-75% of source code
is time constrained

5 = >75% of source code
is time constrained

1 1 1 1 1

Mitre Project Data (page 8 of 18)

Project #24 (A,B,C,D) 24 A B C D

Description of Factors

TURN = Computer Turnaround Time

 .87 = low (interactive)

 1.00 = nom (ART<4hr)

 1.07 = high (ART<12hr)

 1.15 = vhigh (ART>12hr)

1 1 1 1 1

ART = Avg Response Time from job
 submission until results
 are back in developer's
 hands ("logo-to-hardcopy")

Percentage of Source Instructions Developed by Each Access Mode:

Batch = Batch = % of total DSI 0 0 0 0 0

DedP = Dedicated Processor 100 100 100 100 100
 = % of total DSI

TBhp = Test Bed with High
 Priority (% of
 total DSI) 0 0 0 0 0

Int = Interactive
 (% of total DSI) 0 0 0 0 0

P/T = Avg Number of Engineers/
 Programmers per Terminal
 which are readily accessible
 to the development team
 (maximum #) not given

Sites = # of Development Sites
 and when there are
 multiple CSCIs, this #
 should indicate the site
 (coded by #) at which
 each CSCI was developed 1 1 1 1 1

Mitre Project Data (page 9 of 18)

Project #24 (A,B,C,D) 24 A B C D

Description of Factors

TOOL = Use of Software Tools

1.24 = vlow (Basic microprocessor tools)					
1.10 = low (Basic minicomputer tools)					
1.00 = nom (strong minicomputer or basic maxicomputer tools)					
.91 = high (strong maxicomputer tools)	.91	.91	.91	.91	.91
.83 = vhigh (advanced maxicomputer tools)					

AEXP = Application Experience

with Techniques Used	1.13	1.29	1.29	1.29	1.29
1.29 = vlow (<4mos)					
1.13 = low (5 mo - 3 yrs)					
1.00 = nom (3 yrs - 6 yrs)					
.91 = high (6 yrs - 12 yrs)					
.82 = vhigh (>=12 yrs)					

Tech = Avg Experience with

Techniques Used	*	1	1	3	3
1 = 1-4 mos. avg. experience					
2 = 4mos - 1 yr					
3 = 1 - 3 yrs					
4 = 3 - 5 yrs					
5 = >= 6 yrs					

LEXP = Programming Language

Experience	1.14	1.14	1.14	1.14	1.14
1.14 = vlow (<= 1 mo. avg. exp.)					
1.07 = low (4 mos - 1 yr)					
1.00 = nom (1 yr. - 3 yr)					
.95 = high (>= 3 yrs)					

Mitre Project Data (page 10 of 18)

Project #24 (A,B,C,D) 24 A B C D

Description of Factors

VEXP = Virtual Machine
Experience 1.21 1.21 1.21 1.21 1.21
(development and
target computer
hardware, operating
systems and architecture)
1.21 = vlow (<= 1 mo. avg. exp.)
1.10 = low (4 mos. - 1 yr.)
1.00 = nom (1 yr. - 3 yrs)
.90 = high (>= 3 yrs)

SS/T = Average Experience with
Support Software/Tools * 1 1 4 4
1 = < 1 mo.
2 = 1 - 4 mos.
3 = 4 mos. - 1 yr.
4 = 1 yr. - 3 yrs.
5 = 3 yrs. - 6 yrs.

PCAP = Programmer Capability 1 1.17 1 .86 .86
Avg. personnel quality
with respect to overall
industry population.
Based on the programming
team's aptitude for
programming/designing
software, efficiency
and thoroughness, and
ability to communicate
and cooperate.
1.42 = vlow (15th percentile)
1.17 = low (>35th percentile)
1.00 = nom (>55th percentile)
0.86 = high (>75th percentile)
0.70 = vhigh (>90th percentile)

ACAP = Analyst Capability 1 1.19 1 .86 .86
Avg. personnel quality
as described above
1.46 = vlow (15th percentile)
1.19 = low (>35th percentile)
1.00 = nom (>55th percentile)
0.86 = high (>75th percentile)
0.71 = vhigh (>90th percentile)

Mitre Project Data (page 11 of 18)

Project #24 (A,B,C,D) 24 A B C D

Description of Factors

MpAvail = %; Degree to which manpower loading levels are constrained by personnel availability or budget limitations (100% = no manloading constraints) 100 100 100 100 100

PkMload = Peak Software Development Team Manloading over the course of the project (%) 33 15 2 6 11

RELY = Required Software Reliability 1.15 1.15 1.15 1.15 .88
 .75 = vlow
 .88 = low
 1.00 = nom
 1.15 = high
 1.40 = vhigh

CPLX = Software Product Complexity 1.15 1.3 1.15 1.15 .85
 .70 = vlow
 .85 = low
 1.0 = nom
 1.15 = high
 1.30 = vhigh
 1.65 = ehigh

SpecQ = Quality of Specification * 0 1 0 1
 0 = very precise
 1 = precise
 2 = imprecise

DSLOC = Deliverable Lines of Source Code Excluding Documentation 185600 87700 18300 29500 50100

DocLOC = Lines of Source Code Documentation 180400 108200 20000 16000 36200

Mitre Project Data (page 12 of 18)

Project #24 (A,B,C,D) 24 A B C D

Description of Factors

DATA = Data Base Size .94 .94 .94 .94 .94

 .94 = low (D/P<10)

 1.00 = nom (10<D/P<100)

 1.08 = high (100<D/P<1000)

 1.16 = vhigh (D/P>=1000)

D/P = Data Base Size in Bytes or Characters
 LOC

Size Breakdown by Operation as % of LOC

DSR = Data Storage and Retrieval (%)	*	0	5	10	4
OLC = On-Line Communications (%)	*	10	5	0	11
RTC&C = Real-time Command and Control (%)	*	0	0	0	0
IntOp = Interactive Operations	*	35	0	0	12
MathOp = Mathematical Operations (%)	*	55	90	10	31
String = String Manipualtion (%)	*	0	0	80	42
OS = Operating Systems (%)	*	0	0	0	0

Operational Response Requirement as % of LOC

RT = Real-Time (%)	*	55	41	0	16
OL = On-Line (%)	*	35	0	0	0
TC = Time-Constrained (%)	*	10	0	0	84
NTC = Non-Time Critical (%)	*	0	59	100	0

Source Statement Type Mix as % of LOC

Logic = Logical (%)	*	20	20	60	32
Commnd = Command (%)	*	30	10	10	33
Math = Mathmatical (%)	*	40	60	10	31
DatMan = Data Manipualtion (%)	*	0	0	10	0

Mitre Project Data (page 13 of 18)

Project #24 (A,B,C,D) 24 A B C D

Description of Factors

DatDcl = Data Declaration (%) * 10 10 10 0

Display = Special Display
 Requirements * 3 1 not given
 0 = none
 1 = simple input/output
 2 = user friendly
 3 = interactive
 4 = complex requirements/
 severe impact

Languages Used as % of Total Equivalent DSI

HOL = Higher Order Language (%) 77.4 86.4 95.6 1.0 100

ASSMBLY = Assembly Language (%)
 = (100-HOL) 22.6 13.6 4.4 99.0 0.0

HOLang = Higher Order Language
 Used (Name) Jovial " " " "

Reusable Code From Similar Projects

ADPLOC = # of DSLOC adapted
 from existing
 software 155400 72800 15900 24100 42600

Design = % Design Modification
 required (of the
 original design) 3.1 0.0 30.0 0.0 0.0

Code = % Code Modification
 required 2.9 0.0 15.0 0.0 5.0

Integr = % Code Integration
 required 29.9 50.0 50.0 0.0 5.0

Mitre Project Data (page 14 of 18)

Project #24 (A,B,C,D) 24 A B C D

Description of Factors

CPI = Conversion Planning Increment	not given				
0 = None					
1 = Simple conversion schedule, acceptance plan					
2 = Detailed conversion schedule, test and acceptance plans					
3 = Add basic analysis of existing inventory of code and data					
4 = Add detailed inventory, basic documentation of existing system					
5 = Add detailed inventory, detailed documentation of existing system					
DOCUM = Documentation Total (# of pages)	not given				
SFtot = Software Failure History (Total of design and coding errors documented as Software Trouble Reports, Software Problem Reports (Total # of STRs, SPRs, etc)	not given				
IncrDev= Incremental Development (Can, or should an incremental costing approach be used?) (Y/N?)	yes	yes	yes	yes	yes
D&Tcomp = Development and Target Computers (Same or Different?)	same	same	same	same	same
CSCData = CSC Level Data (Is there any?) (Y/N?)	*	yes	yes	yes	yes
SCED = Schedule Constraint	1	1	1	1	1
1.23 = vlow					
1.08 = low					
1.00 = nom					
1.04 = high					
1.10 = vhigh					

Mitre Project Data (page 15 of 18)

Project #24 (A,B,C,D)	24	A	B	C	D
<u>Description of Factors</u>					
EFTpd = Effort expended during the preliminary design phase (# of staff-months)	46.22	13.61	2.74	19.8	10.07
EFTdd = Effort expended during the detailed design phase (# of staff-months)	130.76	60.3	9.66	16.52	44.28
EFTcut = Effort expended during the coding and unit testing phase (# staff-months)	95.66	48.85	10.13	13.69	22.99
EFTit = Effort expended during the integration and testing phase (# of staff-months)	62.54	38.26	5.45	2.18	16.65
EFTtot = Total Effort Expended in software Development (# of staff-months)	335.18	161.02	27.98	52.19	93.99
Eunits = Units in which effort was provided 1 = staff-months 2 = staff-hours					
		2	2	2	2
Convfac = Conversion factor from staff-months to staff-hours (#)	152	152	152	152	152
EPTcal = Normalized total effort (# staff-months)	335.2	161.0	28.0	52.2	94.0

Mitre Project Data (page 16 of 18)

Project #24 (A,B,C,D) 24 A B C D

Description of Factors

Months = Period of time over
 which EFTcal value
 was determined
 (# of months) 26 26 24 23 26

Delta T = Difference between the
 time over which effort
 was expended and the
 duration of the software
 development. Positive
 values indicate that
 EFTcal includes effort
 outside the COCOMO-
 defined software
 development period.
 Negative values indicate
 that the effort data
 is incomplete EDSI
 = Equivalent number
 of Deliverable Source
 Instructions for
 adapted LOC. (#) 0.0 0.0 -2.0 -2.0 0.0

TotEDSI = Total Equivalent
 number of
 Deliverable Source
 Instructions (#) 17207 10920 5009 0 1278

Mode = COCOMO mode of software
 development, as defined
 in table below E E SD SD SD
 O = Organic
 SD = Semi-Detached
 E = Embedded
 FW = Firmware

Concurnt = CSCIs which were
 concurrently
 developed are
 assigned the same
 number (#) * 1 2 3 4

Mitre Project Data (page 17 of 18)

Project #24 (A,B,C,D)

24 A B C D

Description of Factors

COMMENTS = Unusual aspects of programs,
development method, or other
possible cost-impact information.

24 Missing CDAs taken from #19; Effort doesn't incl B-5
generation.

24A Orig. est. 94.4 KDSI; Adapted code from #23

24B Orig. est. 28.6 KDSI; Adapted code from #23

24C Orig. est. 47.3 KDSI; Adapted code from #23

24D Orig. est. 66.4 KDSI; Adapted code from #23

Executive = Customer of the contractor: ESD/OC

PQT-1 = Date of first
Preliminary
Qualification
Test (mm.yy)

* 11.83 10.83 12.83 12.83

Incr-1 = Total DSI accepted
at PQT-1
Project #24

*

PQT-2 = Date of second Preliminary not given
Qualification Test

Incr-2 = Total DSI accepted at PQT-2 not given

PQT-3 = Date of third Preliminary not given
Qualification Test

Incr-3 = Total DSI accepted at PQT-3
(Does not include Incr-1
or Incr-2) not given

PQT-4 = Date of fourth Preliminary
Qualification Test, if any not given

Incr-4 = Total DSI accepted at PQT-4 not given
(does not include Incr-1,
Incr-2, or Incr-3)

Mitre Project Data (page 18 of 18)

Project #24 (A,B,C,D) 24 A B C D

Description of Factors

Dffclty = Putnam's Difficulty Factor;
 This is also the initial
 slope of the Rayleigh
 curve that is fit to the
 manloading curve.
 (4*(total effort)/
 Development Time)2 1.02 1.54 0.39 * 1.47

Product = Productivity = Total
 (new + equivalent
 modified) LOC/
 Calibrated effort
 (from SDR through
 FQT)
 (TEDSI/EFTcal) 141.4 160.4 264.8 103.5 93.4

AAF = Adaptation Adjustment
 Factor = .4 x Design +
 .3 x Code + .3 x Integr 11.1 15.0 31.5 0.0 3.0

CAF = Conversion Adjustment
 Factor = AAF + CPI 11.1 15.0 31.5 0.0 3.0

NewDSI = New lines of code
 developed = Delivered
 LOC - modified LOC 30200 14900 2400 5400 7500

Qeft = Qualify factor for effort
 data Code # 5 5 5 5 5

Qsize = Quality factor for
 TEDSI data Code # 3 3 3 3 3

Date = Midpoint year of the
 development = (date
 of SDR + date of
 FQT)/2 1983 1983 1983 1983 1983

EAF = Effort Adjustment
 factor = Product
 of all COCOMO DEMs
 (#) 2.83 5.09 3.23 2.09 1.18

APPENDIX D:

Intermediate COCOMO Cost Driver Ratings and Effort Multipliers for Project #24

<u>Cost Driver</u>	<u>Rating</u>	<u>Effort Multiplier</u>
RELY	H	1.15
DATA	L	0.94
CPLX	H	1.15
TIME	VH	1.30
STOR	EH	1.56
VIRT	VL	0.87
TURN	NOM	1.00
ACAP	NOM	1.00
AEXP	L	1.13
PCAP	NOM	1.00
VEXP	VL	1.21
LEXP	VL	1.14
MODP	H	0.91
TOOL	H	0.91
SCED	NOM	1.00

=====

EAF = 2.83

APPENDIX E:

PRICE-S (MODE 2) Input Values Project 24 A,B,C,D

ITEM DESCRIPTORS

Platform 1.8 Mgmt Complexity 1.00 External Integ 0.50

COMPONENT 1 titled: PROJECT 24 A

DESCRIPTORS

Internal Integration 0.50 External Integration 0.50
Utilization Fraction 0.75

SCHEDULE

Software Spec Review 982 Preliminary Design Review 0
Critical Design Review 0 Test Readiness Review 0
Functional Config Audit 0

LANGUAGE 1 DESCRIPTORS

Language JOVIAL Source Code 75773
Complexity 1 1.00 Complexity 2 1.00

Non-executable SLOC 0.10 Productivity Factor 4.00

Application Categories	Mix	New Design	New Code
User Defined (APPL=0.00)	0.00	0.00	0.00
Data S/R	0.00	0.00	0.00
Online Comm	0.10	0.17	0.17
Realtime C&C	0.00	0.00	0.00
Interactive	0.35	0.17	0.17
Mathematical	0.55	0.17	0.17
String Manip	0.00	0.00	0.00
Opr Systems	0.00	0.00	0.00

LANGUAGE 2 DESCRIPTORS

Language ASSEMBLY Source Code 11927
Complexity 1 1.00 Complexity 2 1.00

Non-executable SLOC 0.10
Productivity Factor 4.00

Application Categories	Mix	New Design	New Code
User Defined (APPL=0.00)	0.00	0.00	0.00
Data S/R	0.00	0.00	0.00
Online Comm	0.10	0.17	0.17
Realtime C&C	0.00	0.00	0.00
Interactive	0.35	0.17	0.17
Mathematical	0.55	0.17	0.17
String Manip	0.00	0.00	0.00
Opr Systems	0.00	0.00	0.00

PRICE-S (MODE 2) Input Values
Project 24 A,B,C,D

COMPONENT 2 titled: PROJECT 24 B

DESCRIPTORS

Internal Integration	0.50	External Integration	0.50
Utilization Fraction	0.75		

SCHEDULE

Software Spec Review	982	Preliminary Design Review	0
Critical Design Review	0	Test Readiness Review	0
Functional Config Audit	0		

LANGUAGE 1 DESCRIPTORS

Language JOVIAL	Source Code	17495
Complexity 1 1.00	Complexity 2	1.00

Non-executable SLOC 0.10
Productivity Factor 4.00

Application Categories	Mix	New Design	New Code
User Defined (APPL=0.00)	0.00	0.00	0.00
Data S/R	0.05	0.26	0.26
Online Comm	0.05	0.26	0.26
Realtime C&C	0.00	0.00	0.00
Interactive	0.00	0.00	0.00
Mathematical	0.90	0.26	0.26
String Manip	0.00	0.00	0.00
Opr Systems	0.00	0.00	0.00

LANGUAGE 2 DESCRIPTORS

Language ASSEMBLY	Source Code	805
Complexity 1 1.00	Complexity 2	1.00

Non-executable SLOC 0.10
Productivity Factor 4.00

Application Categories	Mix	New Design	New Code
User Defined (APPL=0.00)	0.00	0.00	0.00
Data S/R	0.05	0.26	0.26
Online Comm	0.05	0.26	0.26
Realtime C&C	0.00	0.00	0.00
Interactive	0.00	0.00	0.00
Mathematical	0.90	0.26	0.26
String Manip	0.00	0.00	0.00
Opr Systems	0.00	0.00	0.00

PRICE-S (MODE 2) Input Values
Project 24 A,B,C,D

COMPONENT 3 titled: PROJECT 24 C

DESCRIPTORS

Internal Integration	0.50	External Integration	0.50
Utilization Fraction	0.75		

SCHEDULE

Software Spec Review	982	Preliminary Design Review	0
Critical Design Review	0	Test Readiness Review	0
Functional Config Audit	0		

LANGUAGE 1 DESCRIPTORS

Language ASSEMBLY	Source Code	29500
Complexity 1 1.00	Complexity 2	1.00

Non-executable SLOC 0.10
Productivity Factor 4.00

Application Categories	Mix	New Design	New Code
User Defined (APPL=0.00)	0.00	0.00	0.00
Data S/R	0.10	0.18	0.18
Online Comm	0.00	0.00	0.00
Realtime C&C	0.00	0.00	0.00
Interactive	0.00	0.00	0.00
Mathematical	0.10	0.18	0.18
String Manip	0.80	0.18	0.18
Opr Systems	0.00	0.00	0.00

PRICE-S (MODE 2) Input Values
Project 24 A,B,C,D

COMPONENT 4 titled: PROJECT 24 D

DESCRIPTORS

Internal Integration	0.50	External Integration	0.50
Utilization Fraction	0.50		

SCHEDULE

Software Spec Review	982	Preliminary Design Review	0
Critical Design Review	0	Test Readiness Review	0
Functional Config Audit	0		

LANGUAGE 1 DESCRIPTORS

Language JOVIAL	Source Code	50100
Complexity 1 1.00	Complexity 2	1.00

Non-executable SLOC	0.00
Productivity Factor	4.00

Application Categories	Mix	New Design	New Code
User Defined (APPL=0.00)	0.00	0.00	0.00
Data S/R	0.04	0.19	0.19
Online Comm	0.11	0.19	0.19
Realtime C&C	0.00	0.00	0.00
Interactive	0.12	0.19	0.19
Mathematical	0.31	0.19	0.19
String Manip	0.42	0.19	0.19
Opr Systems	0.00	0.00	0.00

APPENDIX F:

PRICE-S (MODE 2) Estimation Summary

Project 24 A,B,C,D

COSTS IN PERSON MONTHS

	Design	Prog	Data	S/PM	Q/A	Connfig	Total
Sys Concept	0.	0.	0.	0.	0.	0.	0.
Sys/SW Req	0.	0.	0.	0.	0.	0.	0.
SW Requirement	134.	0.	28.	49.	26.	25.	262.
Prelim Design	97.	16.	20.	33.	18.	18.	203.
Detail Design	130.	60.	28.	43.	25.	27.	313.
Code/Test	93.	67.	22.	33.	21.	23.	259.
CSCI Test	145.	146.	52.	68.	48.	55.	514.
System Test	0.	0.	0.	0.	0.	0.	0.
Oper T & E	0.	0.	0.	0.	0.	0.	0.
Sub-Total	599.	289.	150.	226.	138.	148.	1550.
Purchased Cost	-	-	-	-	-	-	0.
TOTAL	-	-	-	-	-	-	1550.

SCHEDULE INFORMATION

Concept Start	SEP 81	TRR	OCT 83 (4.1)
SSR	NOV 81 (3.5)	PCA	AUG 84 (10.0)
SDR	FEB 82 (2.3)	PCA	NOV 84 (3.1)
SSR	SEP 82 (7.2)	PQR	FEB 85 (3.1)
PDR	JAN 83 (4.0)	OTE	OCT 85 (7.7)
CDR	JUN 83 (5.1)		

NOTE: The time frame considered for comparison is highlighted in bold print.

SUPPLEMENTAL INFORMATION

Source Lines of Code	185600
Source Lines of Code/Person Month	144.0

APPENDIX G:

SOFTCOST-R INPUT VALUES

Project #24

SIZING FACTORS

Lines of executable source code:	MAX	AVG	MIN
New:	34.7	30.2	25.6
In existing modules requiring modification:	5.2	4.5	3.8
Deleted from existing modules:	0	0	0
Added to existing modules:	0	0	0
Changed in other ways in existing modules:	0	0	0
Deleted as entire modules	0	0	0
Retested but remained unchanged	0	0	0

Percentage of source code developed that will be delivered = High

MANAGEMENT FACTORS

Phase of the life cycle software work will begin: System Requirements Phase

Percentage of total software requirements:

Well established, stable, and will not change before delivery: 85

Will change slightly before delivery (under baseline control): 15

Will change more drastically before delivery (under baseline control): 0

Complexity of software requirements: High

Expected user involvement in requirements definition: Low

Customer experience in the application area: Low

Customer/implementor organizational interface complexity: Medium

Level of interfaces with other projects or organizations: High

Efficiency of implementing organization: Medium

SOFTCOST-R INPUT VALUES

Project #24

STAFFING FACTORS

Overall personnel qualifications of the team: Low
Percentage of programmers doing design and working development: High
Team's experience with projects of similar size and complexity: low
Average staff experience (in years): 3
Staff experience with:
 the operational computer(s): Low
 the programming languages: Low
 top-down methodology: Medium
 team concepts: Medium
 structured programming: Low

Type of technical reviews held: High

COMPLEXITY FACTORS

Number of different I/O items generated per 1000 lines: Low
Overall complexity of the data base architecture: Low
Complexity of the logical design: Medium
Percentage of the:
 program will be in assembly language: 22
 program will be storage optimized: 90
 program will be timing optimized: 78
 total task will be easy: 84
 total task will be hard: 16

ENVIRONMENTAL FACTORS

Classified security environment for computer: Y
Amount of hardware under concurrent development: Medium
Percent of work done at primary development site: High
Development computer accessibility: High
Development computer availability: High
Software development tools/environment reliability: High
Maturity of system and support software: Medium
Overall adverse constraints on program design: Medium
Percent of program which is real-time and multi-tasking: Low
Software will be adapted to multiple environments: N
Adaptation required to change from development to operational environment: Medium

APPENDIX H:

SOFTCOST-R RESOURCES ESTIMATE

PROJECT #24

Calculated:

Effort (person-months):	126.0
Duration (months):	21.1
Average Staff (persons):	6.0
Productivity (SLEC/person-month):	249.2
Adjusted source lines of executable code (KSLEC):	31.4
Confidence:	10.9%

APPENDIX I:
SPQR/20 INPUT VALUES

Project #24

ESTIMATE AND FINANCIAL INPUTS

Estimate Type: New Program
Estimate Scope: Complete stand-alone program
Estimate Goals: Find the normal average of staff size,
schedules, and quality
Maximum Staff Size: Normal
Minimum Staff Size: Normal
Output Metric: Work Months (Default)
Staff Availability: 100%
Exempt Technical Staff: 100%
Average Work Week: 40 hours
Average Work Year: 220 days
Average Monthly Salary: 5000 (default)
Other Project Costs: 0

PROJECT CLASS: External program developed under Military
Specifications

PROJECT TYPE: Embedded or Realtime program

ENVIRONMENTAL INPUTS

Project Novelty: Functional repeat, but some new features
Office Facilities: Doubled offices and good facilities
Program Requirements: Fairly clear user requirements
Program Design: New designs and partial automated graphics/
text design support
User Documentation: Programmers or users with fully
automated graphics/text support
Response time: One to five second response time in the
norm
Staff Experience: Majority of new hires or novices, with
few experts
Source Code Reusability: Extensive use of reusable code
(>75%)

SPQR/20 INPUT VALUES

Project #24

REUSABLE CODE LANGUAGE

Source Language: Mixed languages
Language Level: 3

REUSABLE CODE SIZE (KLOC): 155.4

REUSABLE FUNCTION POINTS: 1,456

LINES PER FUNCTION POINT: 106.73

NEW PROJECT COMPLEXITY

Logical Complexity: Algorithms and calculations of average complexity

Code Complexity: Fair structure, but some complex modules and paths

Data Complexity: Simple data with few variables and low complexity

NEW CODE SOURCE LANGUAGE

Source Language: Mixed languages
Language Level: 3

NEW CODE SIZE (KLOC): 30.2

APPENDIX J:

SPQR/20 SUMMARY ESTIMATE

PROJECT #24

MODE 1: Normal Average

SECURITY: NONE

START DATE: SEPTEMBER 1982

END DATE: AUGUST 1984

PROJECT DEVELOPMENT ESTIMATE

ACTIVITY	SCHEDULE (MONTHS)	EFFORT (MONTHS)
Planning	1.33	1.33
Requirements	2.08	5.61
Design	8.70	26.83
Coding	10.20	33.76
Integration/Test	8.22	28.18
Documentation	11.05	22.44
Management	23.17	20.40
=====		
Development	30.54	138.55
Overlapped	23.17	
Unpaid Overtime		0.00

APPENDIX K:

System-3 Input Values

Project #24

	Minimum	Nominal	Maximum
DEVELOPER TECHNOLOGY			
APPLication Complexity	2.0	2.0	2.0
Analyst CAPability	3.5	5.5	7.5
Application EXPerience	0.3	0.3	1.0
MODern Practices, use of	7.5	7.5	7.5
Programmer CAPability	3.5	5.5	7.5
TOOL support, automated	7.5	7.5	7.5
TURNaround, logon-hardcopy	2.0	2.0	2.0
ENVIRONMENTAL - COMPUTER			
DISP req'mt, special	0.0	3.2	6.8
MEMory Constraint	4.0	4.0	4.0
TIME Constraint	0.0	0.0	0.0
Real TIME	2.5	5.0	5.0
ENVIRONMENTAL - PRODUCT			
SPECification Level	4.0	6.0	6.0
QUALity Assurance Level	4.0	6.0	6.0
TEST Level	4.0	6.0	6.0
Requirements CHAnGe vol	1.3	1.3	4.0
ReHOSTing develop->target	0.0	0.0	0.0
LANGUage type rating	1.0	1.0	1.0
Language EXPerience	0.5	1.5	2.0
SYSTEM complexity-virtual	2.0	2.0	2.0
System EXPerience-virtual	0.5	1.0	1.0
Virtual Mach. VoLatility	0.0	2.5	2.5
ENVIRONMENTAL - SUPPORT			
MULTiple site development	0.0	0.0	0.0
Resource DEDication	7.0	10.0	10.0
Resource/support LOCation	0.0	0.0	0.0
SIZE & COMPLEXITY SUMMARY			
New Lines of Code	25670	30200	34730
Existing Lines of Code	132090	155400	178710
Lines to be Deleted	0	0	0
Lines to be Modified	3830	4507	5183
Complexity	11.0	11.0	11.0
DEVELOPMENT CONSTRAINTS			
Staffing Rate (Persons/Yr)	0.0	0.0	0.0
Maximum Staff (Persons)	0.0		
Maximum Effort (Person Mos)	0.0		

System-3 Input Values

Project #24

	Minimum	Nominal	Maximum
REUSE - REBUILD IMPACT			
% Design Effort Needed	16.0		
% Implement Eff. Needed	16.0		
% Testing Effort Needed	16.0		
FINANCIAL FACTORS			
Average Staff Pay Rate	11000.0		
Target Schedule	0.0		
% Requirements Effort	8.0		
Req's Eff. Complete @ C/A	0.0		
Req's Schedule Constraint	0.0		
% Integration Effort	29.0		
Avg. Annual Inflation	0.0		

APPENDIX L:

System-3 Summary Report

Project #24

(Minimum Time) Estimate

FULL SCALE DEVELOPMENT

Development Time	20.74	Months
Development Effort	272.69	Person Months
Project Staff, Peak	19.93	Persons
Actual Staffing Rate	19.01	Persons/Year
Productivity	127.28	
Lines/Staff/Month		

REQUIREMENTS AND INTEGRATION

Requirements Time	8.82	Months
Requirements Effort	53.32	Person Months
Total Requirements Cost	584.65	K-Dollars
Integration Time	3.95	Months
Integration Effort	79.08	Person Months

Complexity	11.00
Basic Technology Rating	5730.08
Effective Technology Rating	2279.33

Effective Task Size	29696
Total Task Size	185600

APPENDIX M:

COCOMO Software Development Effort Multipliers (5:118)

	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY	.75	.88	1.00	1.15	1.40	
DATA		.94	1.00	1.08	1.16	
CPLX	.70	.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME			1.00	1.11	1.30	1.66
STOR			1.00	1.06	1.21	1.56
VIRT		.87	1.00	1.15	1.30	
TURN		.87	1.00	1.07	1.15	
Personnel Attributes						
ACAP	1.46	1.19	1.00	.86	.71	
AEXP	1.29	1.13	1.00	.91	.82	
PCAP	1.42	1.17	1.00	.86	.70	
VEXP	1.21	1.10	1.00	.90		
LEXP	1.14	1.07	1.00	.95		
Project Attributes						
MODP	1.24	1.10	1.00	.91	.82	
TOOL	1.24	1.10	1.00	.91	.83	
SCED	1.23	1.08	1.00	1.04	1.10	

Bibliography

1. Abdel-Hamid, T.K. and S.E. Madnick. "An Inegrative Approach to Modeling the Software Management Process: A Basis for Identifying Problems and Evaluating Tools and Techniques," Proceedings of the IEEE Computer Society Workshop on Software Engineering Technology Transfer. 15-23. New York: IEEE Computer Society Press, 1983.
2. Aron, J.D. "Estimating Resources for Large Programming Systems," Proceedings of the Software Engineering Techniques Conference Sponsored by the NATO Science Committee. 262-266. New York: Mason Chaster, 1969.
3. Boddie, John. Crunch Mode: Building Effective Systems on a Tight Schedule. Englewood Cliffs NJ: Prentice-Hall, Inc., 1987.
4. Boehm, Barry W. "Software Engineering Economics," IEEE Transactions on Software Engineering, SE-10: 4-21 (January 1984).
5. Boehm, Barry W. Software Engineering Economics, New Jersey: Prentice-Hall, Inc. 1981.
6. Boehm, Barry W. "Software Life Cycle Factors," Handbook of Software Engineering. Edited by C.R. Vick, Ph.D. and C.V. Ramamoorthy, Ph.D. New York: Van Nostrand Reinhold Company, 1984.
7. Brooks, Frederick P., Jr. The Mythical Man-Month: Essays on Software Engineering. Reading MA: Addison-Wesley, 1975.
8. Bruce, Phillip and Sam M. Pederson. The Software Development Project: Planning and Management. New York: John Wiley and Sons, 1982.
9. Bruggere, Thomas H. "Software Engineering: Management, Personnel and Methodology," Proceedings, Fourth International Conference on Software Engineering. 361-368. New York: IEEE Press, 1979.
10. Cheadle, William G, Manager, Technology Implementation and Support Martin Marietta Astronautics Group. Personal Interview. Martin Marietta Corporation, Denver CO, 22 February 1988.

11. Cheadle, William G. "DOD-STD-2167 Impacts on Software Development," ISPA Journal of Parametrics: 4, December 1986.
12. Commonwealth Books, Inc. New Webster's Vest Pocket Dictionary, Framingham MA: Dennison Manufacturing Company, 1976.
13. Computer Economics, Inc., CEI Presents System-3, Marina del Ray CA: CEI, Inc. 1987.
14. Cooper, Jack. "Software Development Management Planning," IEEE Transactions on Software Engineering, SE-10: 22-26 (January 1984).
15. Daly, Edmund B. "Management of Software Development," IEEE Transactions on Software Engineering, SE-3: 289-299 (May 1977).
16. Department of Defense. Defense System Software Development. Military Standard 2167. Washington: DOD, 27 October 1987.
17. Donelson, William S. "Project Planning and Control," Datamation: 73-80 (June 1976).
18. Doty Associates, Inc. Software Cost Estimation Study, Volume I: Study Results, Final Report. Technical Report No. 151. New York: Rome Air Development Center, February 1977.
19. Driscoll, Alan J., Lt.Col. "Software Visibility and the Program Manager," Defense Systems Management, 1: 48-56. (Spring 1977).
20. Emory, C. William. Business Research Methods. Homewood IL: Irwin, 1985.
21. Ferens, Daniel V. An Introduction to Software Parametric Cost Estimating, Wright-Patterson AFB OH: Air Force Institute of Technology, 1987.
22. Funch, Paul G. Software Cost Data Base. Contract F19628-86-C-0001. Bedford MA: The MITRE Corporation, October 1987.
23. Howes, Norman R. "Managing Software Development Projects for Maximum Productivity," IEEE Transactions On Software Engineering, SE-10: 27-35 (January 1984).

24. Hurst, R.S. "SPMMS-Information Structures In Software Management," Software Engineering Journal, 1: 50-57 (January 1986).
25. Jensen, Randall W. "An Improved Macrolevel Software Development Resource Estimation Model," Fourteenth Asilomar Conference on Circuits, Systems and Computers, Institute of Electrical and Electronic Engineers, New York: 1981
26. Keider, Stephen P. "Why Projects Fail," Datamation: 53-55 (December 1974).
27. Norden, Peter. "Useful Tools for Project Management," Management of Production, edited by M.K. Star. Baltimore: Penguin Books, Inc., 1970.
28. Otte, James E. Handout distributed at the Presentation of PRICE Software Model to ASD Engineers. Aeronautical Systems Division, Air Force Systems Command, Wright-Patterson AFB OH, 1988.
29. Pressman, Roger S. Software Engineering: A Practitioner's Approach. New York: McGraw-Hill Book Company, 1987.
30. Putnam, Lawrence H. and Ray W. Wolverton. "Introduction," Tutorial - Quantitative Management: Software Cost Estimating, edited by Lawrence H. Putnam and Ray W. Wolverton. New York: IEEE Press, 1977.
31. Putnam, Lawrence H. SLIM User's Guide, McLean VA: Quantitative Software Management, Inc., 1980.
32. Putnam, Lawrence H. "The Software Life Cycle: Evidence And Foundation," Tutorial - Quantitative Management: Software Cost Estimating, edited by Lawrence H. Putnam and Ray W. Wolverton. New York: IEEE Press, 1977.
33. Putnam, Lawrence H. "The Software Life Cycle: Practical Application to Estimating Cost, Schedule and Providing Life Cycle Control," Tutorial - Quantitative Management: Software Cost Estimating, edited by Lawrence H. Putnam and Ray W. Wolverton. New York: IEEE Press, 1977.
34. RCA PRICE Systems, RCA PRICE-S Reference Manual, Cherry Hill NJ: RCA, 1987.

35. Reifer, Donald J. "The Software Engineering Checklist," Proceedings of the Computers in Aerospace Conference. 126-128. El Segundo CA: American Institute of Aeronautics and Astronautics, 1977.
36. Reifer, Donald J. Tutorial: Software Management. Los Angeles: IEEE Computer Society Press, 1981.
37. Rook, Paul. "Controlling Software Projects," Software Engineering Journal: Special Issue on Controlling Software Projects, 1: 7-16 (January 1986).
38. Scacchi, Walt. "Managing Software Engineering Projects: A Social Analysis," IEEE Transactions on Software Engineering, SE-10: 49-59 (January 1984).
39. Software Productivity Research, Inc. User Guide: SPOR/20, Cambridge MA: Software Productivity Research, Inc., 1986.
40. Thayer, Richard and Arthur B. Pyster. "Guest Editorial: Software Engineering Management," Software Engineering Journal: Special Issue on Controlling Software Projects, 1: 2 (January 1986).
41. The Analytic Sciences Corporation. The AFSC Cost Estimating Handbook Series. Volume 1, "AFSC Cost Estimating Handbook," Reading MA.
42. Van Patten King, C., J. Bruscano, P. Kane and D. Reifer. SoftCost-R User's Manual, Torrance CA: Reifer Consultants, Inc., 1987.
43. Vogel, Donna. "Possible Software Cost Topics for AFIT Theses." Briefing to Graduate Students in Cost Analysis. Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 19 October 1988.
44. Vosburgh, J., B. Curtis, R. Wolverton, B. Albert, H. Malec, S. Hoben, and Y. Liu. "Productivity Factors and Programing Environments," Proceedings of the 7th International Conference On Software Engineering. 143-152. New York: IEEE Computer Society Press, 1984.
45. Wendt, Major H. and M.W. Evans. "Cost/Schedule Management for Software Development," Proceedings of the 2nd AFSC Standardization Conference. 1053-1069. Dayton OH: Air Force Systems Command, Aeronautical Systems Division, 1982.

46. Wingrove, Alan. "The Problems of Managing Software Projects," Software Engineering Journal: Special Issue on Controlling Software Projects, 1: 3-6 (January 1986).
47. Wolverton, Ray W. "The Cost of Developing Large-Scale Software," Tutorial - Quantitative Management: Software Cost Estimating, edited by Lawrence H. Putnam and Ray W. Wolverton. New York: IEEE Press, 1977.

VITA

Captain Crystal D. Blalock received the degree of Bachelor of Science in Business Administration with Honors from the University of Tennessee at Knoxville in 1983. Her major concentration was in Operations Management. Upon completion of Officer Training School in San Antonio, Texas, she was commissioned as a Second Lieutenant in the USAF on 28 March 1984. She served as a Budget Analyst at the Oklahoma City Air Logistics Center, Tinker AFB, OK and as a Cost Analyst at the Aeronautical System Division, Wright-Patterson AFB, OH prior to entering the School of Systems and Logistics, Graduate Studies in Cost Analysis, Air Force Institute of Technology, in July 1987.



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCA/LSY/88S-2			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION School of Systems and Logistics		6b. OFFICE SYMBOL (if applicable) AFIT/LSQ	7a. NAME OF MONITORING ORGANIZATION		
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433-6583			7b. ADDRESS (City, State, and ZIP Code)		
8a. NAME OF FUNDING / SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
11. TITLE (Include Security Classification) AN ANALYSIS OF SCHEDULE DETERMINATION IN SOFTWARE DEVELOPMENT PROGRAMS AND SOFTWARE DEVELOPMENT ESTIMATION MODELS (UNCLASSIFIED)					
12. PERSONAL AUTHOR(S) Blalock, Crystal D. , B.S., Captain, USAF					
13a. TYPE OF REPORT M.S. Thesis		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Year, Month, Day) 1988 September	
15. PAGE COUNT 159					
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Cost Analysis, Estimates, Costs, Schedule, Computer program, Cost models		
FIELD	GROUP	SUB-GROUP			
05	03				
05	01				
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Thesis Advisor: Daniel V. Ferens Professor of Systems Management Approved for public release IAW AFR 190-1. WILLIAM A. MAVER  17 Oct 88 Associate Dean School of Systems and Logistics Air Force Institute of Technology (AU) Wright-Patterson AFB OH 45433					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Daniel V. Ferens			22b. TELEPHONE (Include Area Code) (513) 255-4845		22c. OFFICE SYMBOL AFIT/LSY

UNCLASSIFIED

Block 19:

Accurate schedule estimation in software development programs is important because delays in the schedule of a software development program can cause delays in the entire schedule of a weapon system.

In order to more accurately predict the schedule of a software development program, estimators need to know which development factors affect schedule. This thesis reports twelve factors identified as heavily influencing software development program schedules. These factors were determined through extensive reviews of literature written by software development experts and from interviews with DOD Program Managers/Engineers and commercial experts who have had experience with software development programs.

Also, there are many commercial software development estimation models on the market today. Five of these models were analyzed for their accuracy in predicting software development programs. The models analyzed were COCOMO, PRICE-S, SOFTCOST-R, SPQR/20, and SYSTEM-3. Inputs to these models were also analyzed for their correlation to schedule prediction.

UNCLASSIFIED