To the Graduate Council:

I am submitting herewith a thesis written by C. Elaine Galbraith entitled, "A Prototype Software Project Management Simulator." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

J.H. Poore, Major Professor

We have read this thesis and
recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's Degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under the rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature C. Elaine Galbraith

Date June 21, 1991

A Prototype Software Project Management Simulator (SPMS)

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

C. Elaine Galbraith

August 1991

Copyright © C. Elaine Galbraith, 1991
All Rights Reserved

ABSTRACT

The Software Project Management Simulator (SPMS) provides a setting for users, typically project managers, to study the effects of their decisions concerning staffing and scheduling of a software development project. The user is given a scenario for developing a hypothetical software project that includes a proposed budget and schedule. To successfully implement the software for the project, the user must select activities in the proper order, assign appropriate staff to these activities, and determine the duration of each activity. The quality of the final software product developed by the user is also important in the simulation and is influenced by scheduling error removal activities such as design and code inspections. The SPMS model is derived from software productivity data and cost estimation models from current literature and includes a data base of productivity factors and algorithms that evaluate the performance of the user.

TABLE OF CONTENTS

Page

CHAPTER 1. STATEMENT OF THE PROBLEM 1

CHAPTER 2. A REVIEW OF RELEVANT LITERATURE 4
- Brooks, *The Mythical Man-Month: Essays on Software Engineering* 4
- Fagan, "Design and Code Inspections to Reduce Errors in Program Development" 5
- Walston and Felix, "A Method of Programming Measurement and Estimation" 7
- Jones, "Measuring Programming Quality and Productivity" 14
- Esterling, "Software Manpower Costs: A Model" 17
- Quinnan, "The Management of Software Engineering Part V: Software Engineering Management Practices" 20
- Boehm, *Software Engineering Economics* 26
- Gaffney, "Approaches to Estimating and Controlling Software Costs" 34
- Jones, *Programming Productivity* 36

CHAPTER 3. APPLYING PRODUCTIVITY FACTORS TO THE MODEL 42
- Creating the Sample Software Product 43
- Designing the Simulator Data Base 50
- Designing the Control Flow and Algorithms 58

CHAPTER 4. OBSERVATIONS FROM USING THE SPMS 69

CHAPTER 5. CONCLUSIONS 74

BIBLIOGRAPHY 77

APPENDICES 79

APPENDIX A: LIFE-CYCLE PHASES AND ACTIVITIES 81

APPENDIX B: SPMS SCREEN AND REPORT LAYOUTS 89

APPENDIX C: SPMS SAMPLE EXECUTION 116

VITA 146

LIST OF TABLES

Page

Table 1: Variables That Correlate Significantly With Programming Productivity 9
Table 2: Maintenance Potential or Undiscovered Defects As A Function of Cumulative Defect Removal Efficiency and Initial Total Defects Per Thousand Lines of Code 17
Table 3: Values for Model Parameters 19
Table 4: Software Life Cycle 23
Table 5: Technical Reviews Within the Software Life Cycle 24
Table 6: Effort and Schedule Estimation Equations 28
Table 7: Phase Distribution of Effort and Schedule: All Modes 30
Table 8: Level Effort Multipliers 31
Table 9: Observed Ranges of Defect Removal Efficiencies For Programming Defect Removal Methods 41
Table 10: Software Equation Estimates 46
Table 11: Schedule and Effort Estimates 70
Table 12: Schedule, Effort, and the Development Constant 71
Table 13: Schedule for a Sample Execution 117

LIST OF FIGURES

Page

Figure 1: The Percent of Effective Working Time Per Day Per Person 21
Figure 2: Relative Project Costs 22
Figure 3: Assumptions of the Simulator 44
Figure 4: COCOMO Schedule and Effort Estimates 45
Figure 5: Example of PERT Chart 49
Figure 6: Relationship of Elements 53
Figure 7: Data Base Elements 54
Figure 8: SPMS Control Flow 59
Figure 9: Determine Employees' Productivity 62
Figure 10: Total Errors Found and Corrected by Phase 67

CHAPTER 1. STATEMENT OF THE PROBLEM

The Software Project Management Simulator (SPMS) provides a setting in which factors that have been strongly correlated to software engineering productivity and schedule can be observed and project managers can learn about the software development process. Most project managers today learn about software engineering productivity from experience, and this experience is very costly. Through simulation of the software development process, project managers can learn by managing the development of a hypothetical software product and observing the effects of their decisions on the final outcome. They can learn which factors most affect programmer productivity and the quality of the software product, thereby becoming better equipped to manage complex software development projects.

There is presently no model that considers every detail of a software development environment when estimating project schedule, cost, and productivity. Many aspects of this complex environment have yet to be quantified. Current models consider the most costly activities in an effort to minimize the risk of overrun in software development. However, budget and schedule overruns occur more frequently than many software developers would like to admit. Although software developers are experienced in creating code, they are not aware of the many elements comprising the software development environment and the effects of these on the software development process, and they are ill prepared to use current models.

Using a cost and schedule model to estimate the resources and time required to develop a software product requires collection of data that is specific to a particular software development environment. This process is sometimes perceived as costly and time-consuming,

with vague rewards that are difficult to sell to a company's upper management. The data required to estimate and manage future software projects, however, can be collected economically. Much of the data can be captured in individual corporate accounting and computer systems. A need exists for creating an awareness of this data and how it can be used to improve the software development process.

The SPMS model is derived from software productivity data and cost estimation models from current literature and includes a data base of productivity factors and algorithms that evaluate the performance of the user. The user is given a scenario for developing a hypothetical software project that includes a proposed budget and schedule. To successfully implement the software for the project, the user must select tasks in the proper order, assign appropriate staff to these tasks, and determine the duration of each task. The quality of the final software product developed by the user is also important in the simulation and is influenced by scheduling error removal tasks such as design and code inspections.

The SPMS provides a setting for users, typically project managers, to study the effects of productivity factors on the schedule and budget of the hypothetical software development project. Throughout the software development process, a project manager is faced with many decisions that will affect the success of a project. Often these decisions are made without a complete understanding of the complex software development environment. Through simulation, project managers can observe the consequences of their decisions and gain an understanding of the effects of these decisions on the quality of the final software product.

The SPMS model combines the work of many professionals in the software industry. In Chapter 2, the relevant results from current literature are presented. The software development life cycles described by Boehm (1981) and Quinnan (1980) provide the basis for the life cycle defined in the SPMS. Productivity factors from Walston and Felix (1977) and

Capers Jones (1986) are included in the data base. Cost and effort models defined by Boehm (1981) and Gaffney (1983) provide the framework for defining the hypothetical software project.

Chapter 3 describes the process of combining the models and data in literature to form the SPMS data base and algorithms. The structure and content of the SPMS data base is shown along with the relationships between the elements of the model. Algorithms that determine the work rate of the project team and the fraction of a week spent productively programming are defined, as are the effects of testing and error removal tasks on the quality of the final software product.

Chapter 4 includes observations that the user must make in determining the correct decisions regarding staffing and task durations. Reports are displayed for each week that passes within the simulation and show the level of productivity achieved, project cost, and the results of inspections and testing. The user must review these reports to determine the project progress.

This thesis concludes with the lessons a user can learn from the SPMS. These lessons include observations such as the productivity of programmers can vary as much as 10 to 1 and removing errors early in the software development process is less costly than removing the same errors later. By creating tools like the SPMS, which allow the study of the process of developing software, many lessons can be learned that can be applied to real software development projects.

The SPMS was written in FoxPro™ and the design and code appear in the technical report "Prototype Software Project Management Simulator (SPMS) Design." The program is complete and operational, but has had very limited testing and use.

CHAPTER 2. A REVIEW OF RELEVANT LITERATURE

In the past 20 years, several papers and books have been published that contribute to the study of programming productivity factors, cost estimation, project management, software quality, and the intricate relationships among these concepts. From a book written by Frederick Brooks in 1975 to the many books and papers published in the 1980's and 1990's, there is much software development information to be studied. The material published today builds on the work of those who began studying the programming environment in the early 1970's. Continued contributions to the study of programming depend on our understanding and application of this initial material, as well as on validation of the principles outlined in these early studies. Summaries of the noteworthy studies that have been considered in the development of the SPMS are included in the following sections.

Brooks, *The Mythical Man-Month: Essays on Software Engineering*

Many books and articles that deal with software productivity data provide information on estimating and measuring productivity in man-months of effort. The concept of a "man-month" must be carefully considered when measuring the effort required to develop software. Frederick P. Brooks (1975) has shown software engineers that men and months are not necessarily interchangeable entities as might be implied in the "man-month" unit of measure. Brooks stresses that if the tasks of a project can be divided in such a way that each task is completely independent of other tasks, then the man-month unit of productivity measurement

can provide a reasonably accurate assessment of project progress. However, many projects cannot be divided in this way. In addition, project team members are forced to interact in order to complete the project, and communication time must be included in the overall project time.

There is also a limit to the number of people who can effectively work together on a project before communication becomes so complex that progress is inhibited. Brooks does not provide a model for project estimation because, at the time of his book, there was not enough empirical data to determine estimation techniques. Brooks does, however, suggest collecting data concerning software development in order to better estimate cost and schedule for future projects.

Brooks takes a critical yet realistic view of every aspect of software development and provides some important lessons learned; his most well-known principle is: "Adding manpower to a late software project makes it later." He has determined that the productivity of programmers can vary as much as 10 to 1, and a programmer may spend only half of a workweek actually writing and debugging code. Brooks also stresses the importance of chief programmer teams and good software development procedures in the success of a project. His work created an awareness in the programming community that led to more study.

Fagan, "Design and Code Inspections to Reduce Errors in Program Development"

M.E. Fagan (1976) outlined a complete inspection process, beginning with the design phase of the software life cycle. At least two inspections are recommended, one after the first design and another after the program is coded and tested by the coder but before it

proceeds through formal unit testing. Fagan strongly advises using inspections to minimize software errors and contends that, if performed properly, these are an economical means of finding errors in programs and their designs. Four people are recommended to serve the inspection roles of moderator, designer, coder, and tester. The details of conducting and tracking the results of the inspections are included in the article, as are significant productivity improvement and error removal data. Significant results from Fagan include:

- Conducting design and code inspections can produce "a 23 percent increase in the productivity of the coding operation alone."
- Finding and correcting errors early in the software development process can be "10 to 100 times less expensive" than finding and correcting them later in the process and also increases productivity over the long run by decreasing rework time.
- Rates of 652 and 539 lines of code per programmer-hour have been observed for design and code inspections of applications programs, respectively.

Work by T.C. Jones (1978), Capers Jones (1986), and Barry Boehm (1981) confirms that the inspections outlined by Fagan provide a means of detecting errors early in the software development process. The number of errors removed by inspections has been observed in many software projects to be from 33% (Myers, 1978) to 89% (Boehm, 1981) of the total presumed errors. This wide variation can be attributed to differing environments and program size. Inspection is a human process and therefore depends heavily on the expertise of the people conducting the inspection.

Walston and Felix, "A Method of Programming Measurement and Estimation"

One journal article concerning programming productivity that is referenced in many current software engineering-related textbooks and technical papers is "A Method of Programming Measurement and Estimation," written by C.E. Walston and C.P. Felix (1977). This article reports on a productivity analysis that was performed by the IBM Federal Systems Division on a base of data from 60 widely varying programming projects. The programs ranged in size from 4,000 to 467,000 lines of code. The total effort ranged from 12 to 11,758 man-months. The programs can be classified as "real-time process control, interactive, report generators, data-base control, and message switching programs." The data base contains the following information.

Programming project data:

- Number of lines of delivered source code ordered by project. (Source lines are 80-character source records provided as input to a language processor. Job control languages, data definitions, link edit language, and comment lines are included. Reused code is not included.)
- Pages of documentation (including program source listings) delivered.
- Source languages used to develop code.

Resource data:

- Total effort (in man-months, including management, administration, analysis, operational support, documentation, design, coding, and testing) required to produce the lines of source code by project.
- Duration of project in months.

Use of improved programming technologies (expressed as a percentage of code developed using each technique):

- Structured programming.
- Top-down development.
- Chief programmer team.
- Design and code inspections.

Derived data:

- Productivity achieved, ordered by project. Programming productivity is defined as the ratio of the delivered source lines of code to the total effort (in man-months) required to produce the lines of code, and is computed from product and resource data.

- Average number of people required to work on the project, computed by dividing the total effort in man-months by the duration of the project in calendar months.

One purpose behind the study of these software development projects was to determine the correlation between the various aspects of the software development environment and productivity, so that an estimator for productivity could be derived. The effort estimator determined from a least-squares fit of the data is $E = 5.2L^{0.91}$, where E is man-months of effort and L is thousands of lines of code (KLOC). This equation indicates that there is an almost linear relationship between the number of lines of code and the effort required to produce the code.

An attempt was then made to determine those factors that are most strongly associated with productivity and to determine a means for automating the project estimation process. To accomplish these objectives, 68 items from the data base were chosen and analyzed for any correlation to productivity. Of these, 29 were shown to be highly correlated to productivity, as is given in Table 1.

The data in Table 1 was determined from line managers' responses to a questionnaire that requested information such as "Customer interface is (less than, equal to, greater than) normal complexity." The first column in the table indicates the factor that was shown to

Table 1: Variables That Correlate Significantly With Programming Productivity

Question or Variable	Response Group Mean Productivity (DSL/MM)			Productivity Change (DSL/MM)
Customer Interface Complexity	<Normal 500	Normal 295	>Normal 124	376
User Participation in the Definition of Requirements	None 491	Some 267	Much 205	286
Customer Originated Program Design Changes	Few 297		Many 196	101
Customer Experience With the Application Area of the Project	None 318	Some 340	Much 206	112
Overall Personnel Experience and Qualifications	Low 132	Average 257	High 410	278
Percentage of Programmers Doing Development Who Participated in the Design of Functional Specifications	<25% 153	25-50% 242	>50% 391	238
Previous Experience With Operational Computer	Minimal 146	Average 270	Extensive 312	166
Previous Experience With Programming Languages	Minimal 122	Average 225	Extensive 385	263
Previous Experience With Application of Similar or Greater Size and Complexity	Minimal 146	Average 221	Extensive 410	264
Ratio of Average Staff Size/Duration (people/month)	<0.5 305	0.5-0.9 310	>0.9 173	132

Table 1: (continued)

Question or Variable	Response Group Mean Productivity (DSL/MM)			Productivity Change (DSL/MM)
Hardware Under Concurrent Development	No 297		Yes 177	120
Development Computer Access, Open Under Special Request	0% 226	1-25% 274	>25% 357	131
Development Computer Access, Closed	0-10% 303	11-85% 251	>85% 170	133
Classified Security Environment for Computer and 25% of Programs and Data	No 289		Yes 156	133
Structured Programming	0-33% 169	34-66% -	66% 301	132
Design and Code Inspections	0-33% 220	34-66% 300	>66% 339	119
Top Down Development	0-33% 196	34-66% 237	>66% 321	125
Chief Programmer Team Usage	0-33% 219	34-66% -	>66% 408	189
Overall Complexity of Code Developed	<Average 314		>Average 185	129
Complexity of Application Processing	<Average 349	Average 345	>Average 168	181
Complexity of Program Flow	<Average 289	Average 299	>Average 209	80
Overall Constraints on Program Design	Minimal 293	Average 286	Severe 166	107

Table 1: (continued)

Question or Variable	Response Group Mean Productivity (DSL/MM)			Productivity Change (DSL/MM)
Program Design Constraints on Storage	Minimal 391	Average 277	Severe 193	198
Program Design Constraints on Timing	Minimal 303	Average 317	Severe 171	132
Code for Real Time or Interactive Operation, or Executing Under Severe Timing Constraint	<10% 279	10-40% 337	>40% 203	76
Percentage of Code for Delivery	0-90% 159	91-99% 327	100% 265	106
Code Classified as Non-Mathematical Application and I/O Formatting Programs	0-33% 188	34-66% 311	67-100% 267	79
Number of Classes of Items in the Data Base per 1000 Lines of Code	0-15 334	16-80 243	>80 193	141
Number of Pages of Delivered Documentation Per 1000 Lines of Delivered Code	0-32 320	33-88 252	>88 195	125

Source: Walston and Felix (1977, pp. 64-65).

affect productivity. The next three columns are the averages, taken across the entire data base, of the "delivered source lines of code per man-month" (DSL/MM) for each category of response. For example, the average productivity for projects in the data base reporting no user participation in the definition of requirements is 491 DSL/MM. The average productivity where the user participated "some" in the requirements definition phase is 267 DSL/MM, and productivity where the user participated "much" is 205 DSL/MM. Each of the factors shown in the table as "overall personnel experience," "code complexity," and "design constraints" is comprised of a combination of responses to more than one question. The article does not indicate which questions from the questionnaire were combined, nor is a copy of the questionnaire included in the article.

A "productivity index" was formulated from the 29 factors listed in the table:

$$I = \sum_{i=1}^{29} W_i X_i$$

where

I = productivity index for a project

W_i = question weight, calculated as one-half $\log_{10}$ of the ratio of total productivity change indicated for a given question i

X_i = question response (+1, 0, -1), depending on whether the response indicates increased, nominal, or decreased productivity

This index was calculated for 51 of the projects in the data base and was graphed against actual productivity. A least-squares fit to the data was performed, and the effort equation previously mentioned, $E = 5.2L^{0.91}$ was derived.

Additional observations were made using the productivity data in the data base. It was found that the number of months (M) required to complete a project of a given number of lines of code (L, measured in KLOC) could be represented by the equation:

$$M = 4.1L^{0.36}.$$

An estimator for the project duration in months can be calculated from the total effort (E) in man-months and is given as:

$$M = 2.47E^{0.35}.$$

The average number of people required to complete a project can be estimated as:

$$S = 0.54E^{0.06},$$

where

S = average number of people on staff and
E = total effort in man-months.

These productivity data and formulas applied to the IBM Federal Systems Division at the time the article was published and may not be accurate in other environments. However, the results illustrate the many productivity factors that can be identified and provide a basis for analyzing other environments. The results also raise the question of how to capture and apply similar data in an effort to predict the outcome of a software development project.

Jones, "Measuring Programming Quality and Productivity"

T.C. Jones (1978) suggests ways to measure programmer productivity and presents the problems associated with each measurement. Jones identifies two divisions in the units of productivity measurement: "work units" and "cost units." Work units are:

- Lines of code written per programmer-month
- Pages of documentation written per writer-month
- CPU hours and connect hours per programmer-month
- Test cases written and executed per programmer-month

Cost units are:

- Programmer-months of effort per thousand lines of code
- CPU hours and connect hours per thousand lines of code
- Dollars expended per thousand lines of code
- Cost per page for documentation
- Cost per defect for maintenance

According to Jones, there are five problems associated with using lines of code as a measure of productivity:

- Sensitivity to line-counting variations
- Ineffectiveness for noncoding tasks
- Tendency to penalize high-level language programs in favor of programs written in Assembler language
- Arithmetic awkwardness in accounting for subtasks
- Attention focusing on the act of coding, which is a misdirection, since the coding of a program is but a small part of the total effort required

By considering "sensitivity to line-counting variations," Jones questions the method of counting lines of code and raises the issue of whether comments, declarations, and statements

should be counted or just executable code. Jones points out that the method of counting code does not matter, but code should be counted consistently and adequately documented within any study.

Jones recognizes that there is some difficulty in using lines of code to represent productivity throughout the software engineering life cycle because many activities, such as requirements analysis and design, do not produce code. In addition, high-level languages appear less productive than low-level languages, yet non-coding tasks take the same amount of time regardless of the programming language used.

By addressing "arithmetic awkwardness," Jones focuses on analyzing development of a completed program in phases, where care should be exercised in calculating the number of lines of code per month in each phase and the numbers used to compare different development projects should be given in the same units of measure.

Using lines of code as a measure of productivity focuses on the act of coding, which is becoming the smallest portion of the overall software development effort. Coding was at one time performed by individual programmers, but now code is written by teams of people performing various analyses and design tasks. Much more effort goes into planning the code than actually writing it. Also, code reuse affects the overall line count for a software product. Because coding is such a small part of a software development project, the focus of measuring productivity has shifted to the entire life cycle. Costs can be associated with each life-cycle element, and progress and costs can be tracked together.

Several results of Jones' analyses are common considerations in software engineering today and include observations such as:

- Expenditures on a software project are higher at the beginning of a project due to the learning curve for requirements analysis, specifications, and functional definition and design.

- Improvements in productivity should lower software development costs.
- The method of measuring productivity in lines of code per month is questionable when code is reused.
- Reuse of code that is structured and modular can be economical.
- Measurement of the cost of defect removal can be deceptive, since high-quality programs will have a higher cost per defect; cost per thousand lines of code is a better measure of the cost of defect removal.
- To determine the total cost of a software project, the costs of the various subtasks should be totaled instead of given in ratios and percentages.

In this article, Jones also provides data on the sources of errors in programming and error removal efficiency. He identifies six sources of errors:

- Functional problems and the misunderstanding of user requests
- Problems of logic and internal program design
- Coding problems
- Documentation problems
- Incorrect repairs or bad fixes
- Miscellaneous causes (a small category)

Typical error data is reported and is shown in Table 2. The first column shows the presumed program errors before inspections or testing. The remaining columns provide figures for errors left in the code after inspections and testing, with several assumed error removal efficiencies. For example, if it is assumed that there are 30 errors in a program and the testing methods are 90% effective, three errors could remain in the program. Jones bases these figures on data he collected, although the actual data is not presented in the article. His results indicate that the possibility exists for from 1.5 to 3.5 errors per KLOC to be left

undetected in a program. Assuming that the underlying assumptions are correct, these data could be used to predict the number of errors occurring in software before testing, and testing could be stopped when the predicted number is approached.

Table 2: Maintenance Potential or Undiscovered Defects As A Function of Cumulative Defect Removal Efficiency and Initial Total Defects Per Thousand Lines of Code

Total Defects Per Thousand Lines of Code	Cumulative Defect Removal Efficiency Percentage					
	90	91	92	93	94	95
35	3.5	3.15	2.8	2.45	2.1	1.75
34	3.4	3.06	2.72	2.38	2.04	1.7
33	3.3	2.97	2.64	2.31	1.98	1.65
32	3.2	2.88	2.56	2.24	1.92	1.6
31	3.1	2.79	2.48	2.17	1.86	1.55
30	3.0	2.7	2.4	2.1	1.8	1.5

Source: Jones (1978, p. 48).

Esterling, "Software Manpower Costs: A Model"

In a *Datamation* article, R. Esterling (1980) proposes a very simple model for determining the effective working time of people on a project. His model considers the effect of overtime and addresses the communication time between the individuals working directly on the project. Management time is considered an overhead activity and is not included in the model, although overhead and other direct costs can be added to the cost of the development effort to obtain a total project cost. This model assumes that no training or

learning time is required for the project staff. The model variables and parameters are defined as:

n = the number of interacting people working directly on the project (managers are not included but are factored into indirect costs),

o = the average number of overtime hours per workday per person,

s = an average personnel base salary per hour,

a = the average fraction of a workday spent on administrative or nondirect work,

t = the average duration of work interruptions (in minutes),

r = the average time (in minutes) needed to regain a train of thought after an interruption,

k = the number of interruptions per workday from people working directly on the project,

p = an average number of interruptions from other sources,

i = indirect costs per person as a fraction of base pay overhead, and

d = differential pay for overtime hours as a fraction of base salary.

Output from the model is:

$$"w = [8 + o - 8a - 4r/60 - p\ (t+r)/60 - k\ (n-1)(t+r)/60]/8,"$$

where

w is the "fraction of useful working time on a project per workday per person, which is closely related to direct labor."

The effective working time calculation represented by the term w begins with the eight-hour workday, adds overtime hours, and subtracts those portions of the day spent on indirect and non-productive tasks such as lunch time. The term "4r" represents the time

required to regain concentration on a task after lunch and two breaks, as well as at the beginning of the workday. The term "/60" is used to convert minutes to hours.

The values that Esterling describes as "average" for programmers are shown in Table 3. Using these values--or values observed in a particular environment--in the equation for w, the productive time spent on a project can be determined. Also, when using w and the model parameters, calculations can be made concerning total time and cost on a project.

Table 3: Values for Model Parameters

			Programmers		
Parameter	Range	Factory Workers	Optimistic	Typical	Pessimistic
a	0-0.5	0.0	0.05	0.10	0.15
t	1-20	3	3	5	10
r	5-10	0.5	0.5	2.0	8.0
k	1-10	1	2	3	4
p	1-10	1	1	4	10
i	1-3	0.2	0.2	0.5	1.02
d	1-2	1.5	1.0	1.0	1.5

Source: Esterling (1980, p. 165).

Esterling proposes the equations:

$$T = 7/(5nw)$$
$$c = n(8+8i+od)s$$
$$e = (nw)/c$$
$$C = 5cT/7 = c/(nw).$$

These equations provide a means for calculating the effective man-days required to complete the project (T), labor cost (c), working time per dollar (e), and the ratio of project cost to man-days (C). The total cost of labor for the project is (nd)(C). By selecting values for the

model parameters and defining a range for the number of people who could perform the work, a determination can be made on the appropriate number of people to assign to a project to minimize cost or time. By choosing various values for the model parameters, the effects of an uninterrupted work environment can also be seen. Figures 1 and 2 show the effect that the number of people has on effective working time and project cost for each of the categories given in Table 3. As can be seen from Figure 1, productivity degrades proportionally as the number of people interacting on a project increases. In addition, Figure 2 shows how project costs increase as the number of required people interfaces expands. Two other interesting observations made by Esterling are: (1) "project completion time can be more than halved by working 60-hour weeks" and (2) "more people on the job are not as detrimental as when no overtime is worked because working efficiency is higher after normal working hours."

Quinnan, "The Management of Software Engineering Part V: Software Engineering Management Practices"

From work done by the IBM Federal Systems Division, R.E. Quinnan (1980) identifies the basic elements of the software engineering life cycle, the products resulting from each element, and the reviews necessary to discover errors as early as possible in the life cycle. Tables 4 and 5 summarize these elements.

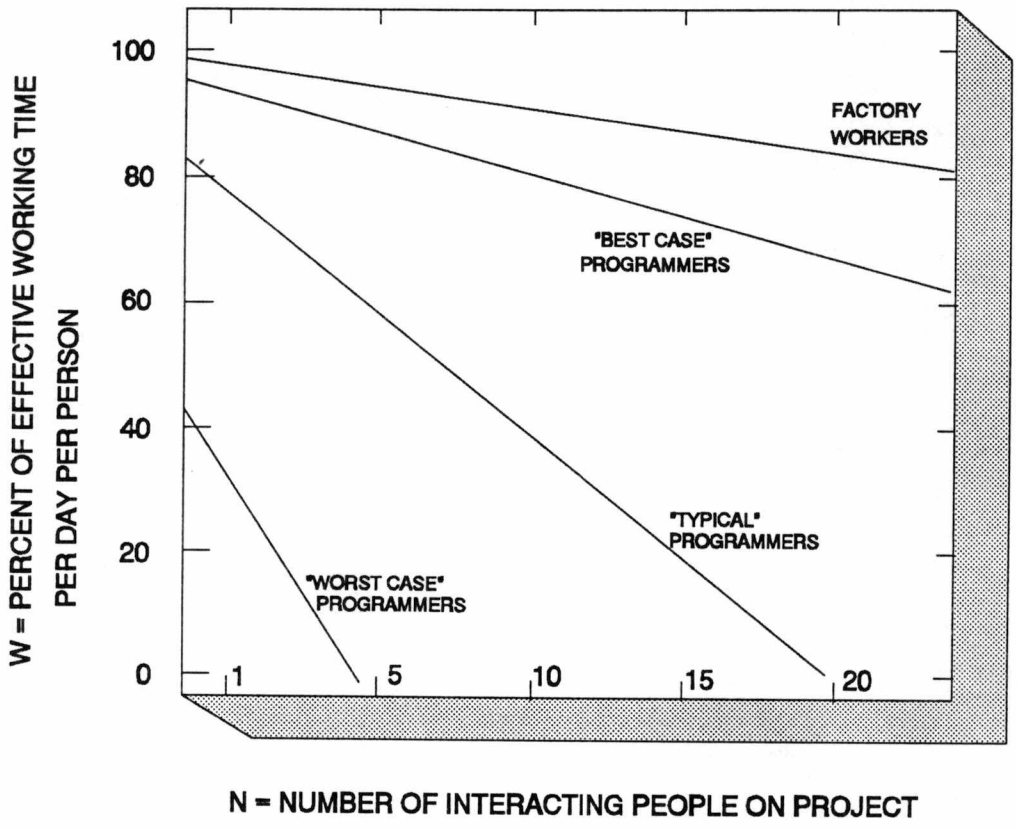


Figure 1: The Percent of Effective Working Time Per Day Per Person

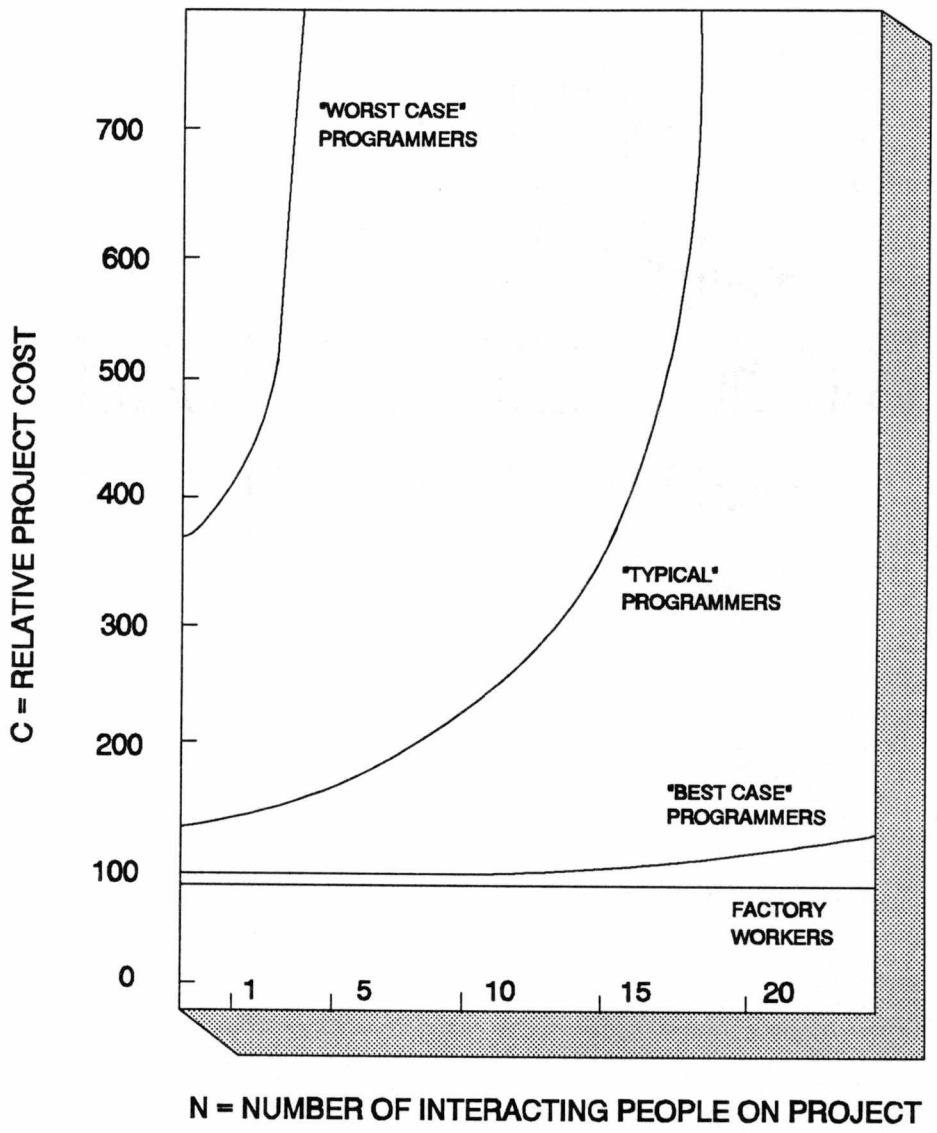


Figure 2: Relative Project Cost

Table 4: Software Life Cycle

Activity	Work Components	Outputs
System Definition	Software Requirements Definition	Software System Requirements Specifications
	Software System Description	Software System Description Document
	Software Development Planning	Software Development Plan
	Engineering Change Analysis	Engineering Change Proposals
Software Design	Functional Design	Functional Design Specifications
	Program Design	Program Design Specifications
	Test Design	Test Design Specifications
	Software Tools	Utilities, Debugging Aids
	Design Evaluation	
Software Development	Module Development	Development (Module) Libraries
	Development Testing	Development Test Procedures/ Reports
	Problem Analysis & Correction	Program Modifications
Software System Test	Software System Test Procedures	Software System Test Procedures
	Software Integration & Test	Software System Integration Library, Test Reports
System/Acceptance	Software Test Support	System Library, Test Reports
	Acceptance Test Support	Delivered Software System/ Acceptance Library
Operational Support	System Operation Support	Level of Effort Assistance, Maintenance
	Training	Level of Effort Training Manuals, Courses
	Site Deployment Support	Level of Effort Assistance
General Support	Project Management	Level of Effort
	Configuration Management/Control	Procedures, Standards, Library Control
	Software Cost Engineering	Cost Management Practice Support
	Quality Assurance	Audits, Quality Assurance Plans
	Administration Centers/Technical Publications	Level of Effort

Source: Quinnan (1980, p. 567).

Table 5: Technical Reviews Within the Software Life Cycle

Software Life-Cycle Activities	Related Work Components	Technical Reviews
System Definition	Software Requirements Definition Software System Description	System Requirements Software System Specifications Test Plan
	Software Development Planning	Documentation Outline
Software Design	Functional Design	Software System Design Integration Plan
	Program Design	Module Design
	Test Design	Test Plan Test Specification
	Software Tools	
	Design Evaluation	
Software Development	Module Development	Module Implementation Unit Test Procedure
	Development Testing	Module Qualification Test Procedure
Software System Test	Software System Test Procedures System Integration and Test	Test Procedures Software System Qualification
System & Acceptance Test	System Test Support Acceptance Test Support	Software System Acceptance Documentation Completion
Operational Support	System Operation Support Training Site Deployment Support	

Source: Quinnan (1980, p. 468).

Quinnan outlines a "software engineering management model" that considers "technical review," "cost management," and "software program management." These concepts combine to create an environment in which the software engineering process can be managed effectively and also produce the best product possible.

Creating a software product involves iterations within each of the activities and the corresponding work components, outputs, and reviews identified in these tables. In the course of development, there may be several software design solutions that, after "technical review," are either flawed or not cost-effective and must be changed. Therefore, the system definition may be again reviewed and a new design created. "Cost management" involves cyclic reviews of projected versus actual expenses, along with reviews of projected and actual software development progress, so that a cost can be associated with each life-cycle activity. Quinnan refers to this practice as "design-to-cost." Design and cost estimation efforts are coordinated in order to more accurately estimate and control project expenses.

While technical review and cost management deal with day-to-day activities in managing the software development process, "software program management" coordinates the various organizations involved in this process. In Quinnan's model, these organizations may be departments within a company or subcontractors hired for specific tasks. The program manager must clearly define all tasks to be performed and coordinate them with the performers. A software development plan defines the tasks and schedule and is used to track total project progress. It is updated each month with accomplishments, progress, and the next month's plans. The participating organizations regularly report this information to the program manager.

Quinnan asserts that software program management facilitates communication between the many participants in the software development process. The technical reviews

provide a link between the technical design and project and program management. Through program management, technical reviews, and design-to-cost practices, software development can be effectively managed and controlled.

Boehm, *Software Engineering Economics*

The Constructive Cost Model (COCOMO) was developed by Barry Boehm, and its usage is fully explained in *Software Engineering Economics* (1981). This model is considered the best cost estimation model in use today because it considers certain requirements and environmental factors that other models do not. Some of the productivity factors discovered in the previously described article by Walston and Felix are incorporated into the COCOMO model. Utilizing COCOMO permits study of the effects of these factors on a project's schedule and cost, thereby providing an aid to decision-making.

Like several other models, COCOMO uses the product size in thousands of lines of code as a basis for estimation, which Boehm expresses in "thousand delivered source instructions" (KDSI). The model can be used in three levels of detail, depending on the accuracy required for the estimate and the information available on a software development project. The Basic COCOMO model is simple to use and can provide the rough estimate needed at the beginning of a project, when only the approximate size of the project is known. The Intermediate model employs productivity data and software requirements to provide a more accurate estimate, and it can be used after the requirements phase. The Detailed model considers more of the subtleties of the software engineering process and may be more applicable to large software development projects than are the other versions of the model.

For all three versions of the COCOMO model, Boehm followed an incremental development process to determine the following life-cycle phases:

1. system feasibility,
2. software plans and requirements,
3. product design,
4. detailed design and verification,
5. code and unit testing,
6. integration and product verification,
7. implementation and system testing, and
8. operations and maintenance and revalidation.

The software development project is divided into increments that are functionally independent, and each increment of a product is developed following the activities in phases 4-8. Other important assumptions common to each version of the model are:

1. the thousand lines of code (KDSI) estimated for a project do not include comments;

2. the life-cycle phases from design to the end of integration (3-6) are considered together in the estimation process (all other phases can be estimated separately);

3. the cost associated with support personnel is not included (only direct labor charges are included);

4. a man-month is 152 hours;

5. non-productive time is relatively small, with the attitude of the project team being good and all doing their best to be productive;

6. requirements remain unchanged for a single estimate (if requirements do change, another estimate is made with the new requirements);

7. the influence of the cost drivers is phase dependent; and

8. certain techniques such as code inspections, walkthroughs, and proper documentation are used to ensure the quality of the developed software in every phase of the software life cycle.

There are three types of projects considered by Boehm in these estimation models: organic, semi-detached, and embedded. Each type uses different estimation equations, as shown in Table 6.

Table 6: Effort and Schedule Estimation Equations

	Effort	Schedule
Basic COCOMO:		
Organic	$MM = 2.4\ (KDSI)^{1.05}$	$TDEV = 2.5\ (MM)^{.38}$
Semi-Detached	$MM = 3.0\ (KDSI)^{1.12}$	$TDEV = 2.5\ (MM)^{.35}$
Embedded	$MM = 3.6\ (KDSI)^{1.20}$	$TDEV = 2.5\ (MM)^{.32}$
Intermediate and Detailed COCOMO:		
Organic	$MM = 3.2\ (KDSI)^{1.05}$	$TDEV = 2.5\ (MM)^{.38}$
Semi-Detached	$MM = 3.0\ (KDSI)^{1.12}$	$TDEV = 2.5\ (MM)^{.35}$
Embedded	$MM = 2.8\ (KDSI)^{1.20}$	$TDEV = 2.5\ (MM)^{.32}$

Information taken from Boehm (1981, pp. 75 and 117).

An "organic mode" project is typically less than 50 KDSI and is developed in an environment familiar enough to the personnel that no training is required. The program logic is rather simple and straightforward. A reporting system would be an example of an organic mode project. A "semi-detached" project is characterized by a size of up to 300 KDSI. A mixture of personnel experience and inexperience, as well as a lack of familiarity with some technical aspects of the project, are assumed. Simulators are typically semi-detached projects. An "embedded mode" project is one that has constraints on hardware and software and that may be governed by regulations and procedures particular to the client. A project of this type may be of any size. Critical software that operates medical machinery or instruments is

considered to be embedded software. Obviously, the more constraints placed on a software project, the longer it takes to develop it to the specifications, as is reflected in the effort equations in Table 6.

To aid in determining the effort and schedule for each phase of a project, Boehm suggests use of the data in Table 7. These data can be used to estimate the number of people required on a project and the duration of each phase. The calculations are simple, and the values for projects that are between the sizes given in the table can be calculated using linear interpolation. Calculations for the number of people required and for cost can also be performed simply for any or all phases of a project. The labor cost of a project can be estimated by multiplying the total man-months by the cost per man-month. This is true for all three COCOMO models.

The Detailed COCOMO model decomposes the project into three levels (module, subsystem, and system), and it applies a group of "effort multipliers" to the module and subsystem levels. The multipliers take into consideration that some factors have a greater effect in some phases than in others.

Review of the values of the multipliers in Table 8 reveals the effects of each factor on each phase of a project. Table 8 does not include multipliers for the system level, as system-level estimates are determined from module and subsystem estimates. The procedure for using the Detailed model to perform estimates involves dividing a system into its subsystem components and module components and estimating the size of each component. Four of the productivity factors considered in the model are applicable to the module level, and 11 apply to the subsystem level, as shown in Table 8.

Table 7: Phase Distribution of Effort and Schedule: All Modes

Distribution Mode	Phase	Size: Small 2 KDSI	Inter-mediate 8 KDSI	Medium 32 KDSI	Large 128 KDSI	Very Large 512 KDSI
Effort						
Organic	Plans & Requirements (%)	6	6	6	6	
	Product Design	16	16	16	16	
	Programming	68	65	62	59	
	Detailed Design	26	25	24	23	
	Code and Unit Test	42	40	38	36	
	Integration & Test	16	19	22	25	
Semidetached	Plans & Requirements (%)	7	7	7	7	7
	Product Design	17	17	17	17	17
	Programming	64	61	58	55	52
	Detailed Design	27	26	25	24	23
	Code and Unit Test	37	35	33	31	29
	Integration & Test	19	22	25	28	31
Embedded	Plans & Requirements (%)	8	8	8	8	8
	Product Design	18	18	18	18	18
	Programming	60	57	54	51	48
	Detailed Design	28	27	26	25	24
	Code and Unit Test	32	30	28	26	24
	Integration & Test	22	25	28	31	34
Schedule						
Organic	Plans & Requirements (%)	10	11	12	13	
	Product Design	19	19	19	19	
	Programming	63	59	55	51	
	Integration & Test	18	22	26	30	
Semidetached	Plans & Requirements (%)	16	18	20	22	24
	Product Design	24	25	26	27	28
	Programming	56	52	48	44	40
	Integration & Test	20	23	26	29	32
Embedded	Plans & Requirements (%)	24	28	32	36	40
	Product Design	30	32	34	36	38
	Programming	48	44	40	36	32
	Integration & Test	22	24	26	28	30

Source: Boehm (1981, p. 90).

Table 8: Level Effort Multipliers

Attribute	Rating	RPD	DD	CUT	IT
Module Level Effort Multipliers					
	Very Low	.70	.70	.70	.70
	Low	.85	.85	.85	.85
CPLX	Nominal	1.00	1.00	1.00	1.00
	High	1.15	1.15	1.15	1.15
	Very High	1.30	1.30	1.30	1.30
	Extra High	1.65	1.65	1.65	1.65
	Very Low	1.00	1.50	1.50	1.50
	Low	1.00	1.20	1.20	1.20
PCAP	Nominal	1.00	1.00	1.00	1.00
	High	1.00	.83	.83	.83
	Very High	1.00	.65	.65	.65
	Very Low	1.10	1.10	1.30	1.30
	Low	1.05	1.05	1.15	1.15
VEXP	Nominal	1.00	1.00	1.00	1.00
	High	.90	.90	.90	.90
	Very Low	1.02	1.10	1.20	1.20
	Low	1.00	1.05	1.10	1.10
LEXP	Nominal	1.00	1.00	1.00	1.00
	High	1.00	.98	.92	.92
Subsystem Level Effort Multipliers					
Product					
	Very Low	.80	.80	.80	.60
	Low	.90	.90	.90	.80
RELY	Nominal	1.00	1.00	1.00	1.00
	High	1.10	1.10	1.10	1.30
	Very High	1.30	1.30	1.30	1.70

Product Attributes:
RELY = Required Software Reliability
DATA = Data Base Size
CPLX = Product Complexity

Personnel Attributes:
ACAP = Analyst Capability
AEXP = Applications Experience
PCAP = Programmer Capability
VEXP = Virtual Machine Exp.
LEXP = Programming Language Exp.

Project Attributes:
MODP = Use of Modern Prog. Practices
TOOL = Use of Software Tools
SCED = Required Development Schedule

Computer Attributes:
TIME = Execution Time Constraint
STOR = Main Storage Constraint
VIRT = Virtual Machine Volatility
TURN = Computer Turnaround Time

Table 8: (continued)

Attribute	Rating	RPD	DD	CUT	IT
Subsystem Level Effort Multipliers (continued)					
	Low	.95	.95	.95	.90
DATA	Nominal	1.00	1.00	1.00	1.00
	High	1.10	1.05	1.05	1.15
	Very High	1.20	1.10	1.10	1.30
Computer					
	Nominal	1.00	1.00	1.00	1.00
TIME	High	1.10	1.10	1.10	1.15
	Very High	1.30	1.25	1.25	1.40
	Extra High	1.65	1.55	1.55	1.95
	Nominal	1.00	1.00	1.00	1.00
STOR	High	1.05	1.05	1.05	1.10
	Very High	1.20	1.15	1.15	1.35
	Extra High	1.55	1.45	1.45	1.85
	Low	.95	.90	.85	.80
	Nominal	1.00	1.00	1.00	1.00
VIRT	High	1.10	1.12	1.10	1.15
	Very High	1.20	1.25	1.30	1.40
	Low	.98	.95	.70	.90
	Nominal	1.00	1.00	1.00	1.00
TURN	High	1.00	1.00	1.10	1.15
	Very High	1.02	1.05	1.20	1.30
Personnel					
	Very Low	1.80	1.35	1.35	1.50
	Low	1.35	1.15	1.15	1.20
ACAP	Nominal	1.00	1.00	1.00	1.00
	High	.75	.90	.90	.85
	Very High	.55	.75	.75	.70

Product Attributes:
RELY = Required Software Reliability
DATA = Data Base Size
CPLX = Product Complexity

Personnel Attributes:
ACAP = Analyst Capability
AEXP = Applications Experience
PCAP = Programmer Capability
VEXP = Virtual Machine Exp.
LEXP = Programming Language Exp.

Project Attributes:
MODP = Use of Modern Prog. Practices
TOOL = Use of Software Tools
SCED = Required Development Schedule

Computer Attributes:
TIME = Execution Time Constraint
STOR = Main Storage Constraint
VIRT = Virtual Machine Volatility
TURN = Computer Turnaround Time

Table 8: (continued)

	Attribute	Rating	RPD	DD	CUT	IT
Subsystem Level Effort Multipliers (continued)						
		Very Low	1.40	1.30	1.25	1.25
		Low	1.00	1.15	1.10	1.10
	AEXP	Nominal	1.00	1.00	1.00	1.00
		High	.87	.90	.92	.92
		Very High	.75	.80	.85	.85
Project						
		Very Low	1.05	1.10	1.25	1.50
		Low	1.00	1.05	1.10	1.20
	MODP	Nominal	1.00	1.00	1.00	1.00
		High	1.00	.95	.90	.83
		Very High	1.00	.90	.80	.65
		Very Low	1.02	1.05	1.35	1.45
		Low	1.00	1.02	1.15	1.20
	TOOL	Nominal	1.00	1.00	1.00	1.00
		High	.98	.95	.90	.85
		Very High	.95	.90	.80	.70
		Very Low	1.10	1.25	1.25	1.25
		Low	1.00	1.15	1.15	1.10
	SCED	Nominal	1.00	1.00	1.00	1.00
		High	1.10	1.10	1.00	1.00
		Very High	1.15	1.15	1.05	1.05

Product Attributes:
RELY = Required Software Reliability
DATA = Data Base Size
CPLX = Product Complexity

Personnel Attributes:
ACAP = Analyst Capability
AEXP = Applications Experience
PCAP = Programmer Capability
VEXP = Virtual Machine Exp.
LEXP = Programming Language Exp.

Project Attributes:
MODP = Use of Modern Prog. Practices
TOOL = Use of Software Tools
SCED = Required Development Schedule

Computer Attributes:
TIME = Execution Time Constraint
STOR = Main Storage Constraint
VIRT = Virtual Machine Volatility
TURN = Computer Turnaround Time

Source: Boehm (1981, p. 354-355).

From simple calculations, the various aspects of a project and of productivity can be analyzed. The COCOMO model provides a means for determining the effects of productivity factors on system development cost. The man-months of effort for development can be multiplied by the cost per man-month to determine the total cost, cost per phase, and cost per subsystem or module.

The productivity factors included in Boehm's COCOMO model are the ones that have been most highly correlated to productivity. Factors such as the number of pages of documentation, personnel continuity, management techniques, and customer interface are not included in the model. For some environments, factors such as these may be highly correlated to productivity and should be included in a cost model for that environment. As with any cost model that considers productivity, there will be errors in the estimates incurred by the different environments. When attempting to estimate system development costs, it pays to be aware of the software development environment.

Gaffney, "Approaches to Estimating and Controlling Software Costs"

John Gaffney (1983) defines a model that provides software development effort estimation and schedule/effort tradeoffs using the life-cycle elements given by Quinnan (1980) and the work of Putnam (1978). The process outlined by Gaffney involves estimating the size and effort in lines of code and person-months, respectively, of each work component that pertains to software development, and then summing the estimates to determine the total software development effort. Also, the rate at which code is produced for each work

component must be known in person-months per thousand lines of code (PM/KLOC). The total labor effort is given by:

Total labor (man-months) =

$$\sum_{i=1}^{\eta} pe_i * S = K$$

Where:

η	=	number of work components
pe_i	=	work rate #i
S	=	amount of source lines of code (KSLOC)
K	=	man-months of effort

Using the above effort estimation as a baseline, Gaffney proposes that using Putnam's Software Equation for individual life-cycle elements aids in refining the estimate and determining an appropriate relationship between the software product size, labor effort, and schedule. The particular software equation that was developed from the IBM Federal Systems Division data is:

$$"S = C * k^{.6288} * t^{.555}"$$

where

S is the number of source lines of code delivered,

C is a constant that reflects the development environment and software complexity,

k is the effort in person-years, and

t is the development time in years.

From the IBM data, C was computed to be between 3700 and 6100 for various software products. By plugging in known values for S, C, and either k or t in the above equation, the

person-years or the calendar years required to develop a product of a specific size in a given environment can be calculated. Effort and schedule for new projects can be estimated by comparing S, C, k, and t with the same data for similar completed projects. Also, the effect of schedule compression on required labor can be observed by plugging in various values for t and solving for k.

Jones, *Programming Productivity*

Capers Jones (1986) gives three measurements of productivity: time, cost, and yield. There are no common units for these measurements, therefore they are not comparable and cannot be easily combined in calculations without caution. Time is measured in pages of documentation or lines of code per unit of time, cost is measured in dollars, and yield is measured in pages of documentation or lines of code per function and the number of defects or repairs. Jones suggests using a measure of "functionality per unit of labor" instead of lines of code, although he does not provide adequate information on this unit of measure to make it practical at present. Due to the availability of previously gathered data, lines of code are still given as a productivity measure in the data he reports. He thoroughly documents the line-counting methods and the conditions under which code is developed.

Inherent in the problem of estimating the time for and cost of developing software is identifying all activities in the development life cycle. A detailed cost model should consider all activities and their duration. Jones suggests the following "peripheral and support activities" in addition to code development:

- Documentation
- Defect removal
- Travel and communication
- Hiring and relocation
- Capital equipment
- Quality assurance
- Training and education
- Integration
- Canceled projects
- Measurement and tracking

In a set of case studies reported in *Programming Productivity*, Jones identifies the following "software development activities."

- Management and planning
- Systems analysis and design
- Programming new code
- Analysis of existing code for enhancements (when this occurs)
- Acquisition of reusable code (when available)
- Internal documentation creation
  - Cost estimates and budgets
  - Development plans
  - Initial and final external specifications
  - Detailed module specifications
  - Inspection and test plans
  - Fault reports and fault responses
- External documentation creation
  - Installation guide
  - Screen and menu creation
  - User's guide
  - Maintenance Manual
- Document production and distribution
- Design and code inspections (unless specifically excluded)
- Integration of code and library control
- Unit, function, and system testing
- Defect repairs by development personnel
- Normal project meetings

These activities do not include maintenance after the program is developed. Jones suggests that these activities "only account for somewhere between 60% and 80% of the probable

effort that will go into a programming system." Project-specific activities account for the remaining effort.

Excluded from the Jones model are activities such as:

- bidding and contract negotiations,
- customer or user involvement in the project,
- analysis and selection of tools and support packages,
- tool customizing and systems programming support,
- development and control of reusable code libraries,
- formal quality assurance,
- formal project auditing,
- hiring and staffing expenses and effort,
- subcontract negotiations and contract programming,
- on-the-job or classroom education of project personnel,
- development of customer classes and textbooks, and
- travel expenses during development.

Overhead activities (accounting, marketing, personnel, etc.) and certain maintenance activities (training users) are also excluded from the model.

Jones identifies "20 factors whose quantified impacts on software projects have been determined from historical data":

1. The programming languages used
2. Program size
3. The experience of programmers and design personnel
4. The novelty of the requirements
5. The complexity of the program and its data
6. The use of structured programming methods
7. Program class or the distribution method
8. Program type or the application area
9. Tools and environmental conditions
10. Enhancing existing programs and systems
11. Maintaining existing programs and systems
12. Reusing existing modules and standard designs
13. Program generators
14. Fourth-generation languages
15. Geographic separation of development locations
16. Defect potentials and removal methods
17. Documentation

18. Prototyping before main development begins
19. Project teams and organization structures
20. Morale and compensation of staff

Twenty-five "intangible" factors identified by Jones for which there is little or no data are:

1. Schedule and resource constraints
2. Unpaid overtime
3. Staff size
4. Total enterprise size
5. Attrition during development
6. Hiring and relocation during development
7. Business systems and/or strategic planning
8. User participation in requirements and design
9. End-user development
10. Information centers
11. Development centers
12. Staff training and education
13. Standards and formal development methods
14. Canceled projects and software disasters
15. Project redirection and restarts
16. Project transfers from city to city
17. Response time and computer facilities
18. Physical facilities and office space
19. Acquiring and modifying purchased software
20. Internal politics and power struggles
21. Legal and statutory constraints
22. U.S. export license requirements
23. Enterprise policies and practices
24. Measuring productivity and quality
25. Productivity and quality improvement steps

Jones contends that his 20 tangible factors "appear to account for more than 90% of the real variations in productivity that occur in industry." Most of these factors remain constant in a single development project but may vary across different projects making project comparisons difficult.

In the study of defect potentials and defect removal methods, Jones categorizes the types of software defects as: requirements, design, coding, documentation, administrative, and bad fixes. These categories correspond to software life-cycle phases and the business of

producing software and show the sources of errors in software. The methods for reducing errors in each phase are different, and therefore should be handled separately. Jones identifies over 40 defect removal methods that can be applied in the software engineering process and points out that there are no conclusive studies that deal with the cost and efficiencies of all of these methods. There are, however, many observations from programmers and project managers as to which methods seem to produce good results. The study by Fagan (1976) also considers some of these methods and their efficiencies.

There is general agreement in the software engineering community that the requirements and design phases are error-prone, and that errors not found early can cause serious problems in later phases. However, the research in this area is sparse and there are no proven methods for solving the problem. Many questions remain on how to determine when requirements are complete and how to present requirements to users in an understandable way and thereby ensure correctness. Rapid prototyping has proven helpful in discovering requirements early on and may remove over 65% of requirements and design errors, while relying on people to check their own work may flush out less than 30% of the errors. Reviews by a quality assurance team have not proven very effective. Jones suggests using formal design inspections, rapid prototyping, and informal reviews to effectively remove errors in the development process. He also writes that "testing alone with no reviews, models, or prototypes is invariably inefficient and is usually expensive as well." Often, the early development phases are not properly completed in order to get to testing quicker, delaying error detection and causing costly design and code rewrites.

Typical error removal consists of unit, function, integration, and system testing. According to Jones, the efficiency of each type of testing may be less than 50% and may be less than 75% if all of these types of testing are applied. Table 9 shows the efficiencies of

several removal methods that have been observed but that have not been proven in controlled studies.

Jones notes that a "complete series" of tests should include the most efficient tests for a phase and consider every phase of development. When considering total cost, schedule, and the quality of the system developed, his data indicates that a series lacking in design reviews predisposes the development process to a higher and more costly rate of error in the integration and testing phases: "Projects that aim from the beginning at achieving the shortest possible schedules, regardless of quality considerations, tend to have fairly high frequencies of both schedule and cost overruns. Software projects that aim initially at achieving the highest possible levels of quality and reliability tend to have the best schedule adherence records, the highest productivity, and even the best marketplace success."

Table 9: Observed Ranges of Defect Removal Efficiencies For Programming Defect Removal Methods

Removal Step	Lowest Efficiency	Modal Efficiency	Highest Efficiency
Personal Checking of Design or Documents	15%	35%	70%
Informal Group Design Reviews	30%	40%	60%
Formal Design Inspections	35%	55%	75%
Formal Code Inspections	30%	60%	70%
Modeling or Prototyping	35%	65%	80%
Personal Desk Checking of Code	20%	40%	60%
Unit Testing (Single Modules)	10%	25%	50%
Function Testing (Related Modules)	20%	35%	55%
Integration Testing (Complete System)	25%	45%	60%
Field Testing (Live Data)	35%	50%	65%
Cumulative Efficiency of Complete Series	93%	99%	99%

Source: Jones (1986, p. 179).

CHAPTER 3. APPLYING PRODUCTIVITY FACTORS TO THE MODEL

The observations and empirical data presented in the previous chapter were integrated into a software system that simulates the software project development process. Life-cycle elements and associated project estimation data, productivity factors and development rate data, effective working time calculations, design and code inspection data, and testing and error removal data were combined from the various sources described in Chapter 2 to create an environment for developing a sample software project. The system allows a user to select tasks from a given list of life-cycle elements, assign start and end dates, select analysts and programmers to perform the tasks, and receive project management reports that track productivity and cost by task or for the entire project. The SPMS consists of an "ideal" model that is used to evaluate the user's performance in completing the software development project. The model enforces a particular life cycle that is unknown to the user, thus creating a game-like environment.

Developing the SPMS involved creating a sample software product that the user must match against the underlying model at all phases, from gathering the requirements through acceptance testing. It also involved designing a base of data and decision-making algorithms from which to judge the user's performance in completing the development of the product. The user is shown detailed progress and productivity reports to aid decision making. The life cycles described by Boehm (1981) and Quinnan (1980) and the corresponding cost model described by Boehm were used as guidelines for identifying phases, activities, and tasks in the life cycle and effort and schedule estimates for the number of lines of code to be written by the programmers in the simulation. These estimates were verified against Gaffney's model

(1983) and found to be consistent. Gaffney's work also provided a method of varying the schedule and staffing to study possible simulation outcomes. The data base and algorithms were designed by combining the data and observations found in literature with the experience of the author in managing and implementing software systems.

Creating the Sample Software Product

In order to create an environment in which the user could actually observe the progress of a software development effort from a project manager's perspective, the product to be developed had to be defined so that the productivity factors included would be consistent with those observed by experience or reported in literature. The resources and ideal schedule required to successfully complete the product had to be consistent with the size of the finished product. Because the data reported in literature represents a diverse set of project development environments, its applicability was limited. For this reason, assumptions were made about the sample software project's development environment, and these assumptions play a significant role in the resource and schedule estimation of this product.

The Detailed COCOMO model was used to determine the effort and schedule for a 16,500-line software product. Because the COCOMO model was used, its assumptions must be included in the description of the development environment for the SPMS. A list of the assumptions that directly affect the user of the SPMS is shown in Figure 3. Other detailed assumptions concerning the COCOMO model can be found in the work by Boehm (1981). The results of COCOMO model estimation are shown in Figure 4. The SPMS model uses these results as a minimum requirement for judging the performance of the user. Any

1. The Waterfall life cycle is implemented.
2. Staffing estimates include individuals contributing directly to the project (programmers, analysts, program librarians, and project manager) and exclude support staff.
3. The cost of support staff is included in overhead calculations.
4. Company overhead is 80% of labor cost.
5. A contract fee is negotiated to be 15% of labor cost.
6. A person-month is 152 hours.
7. Individuals attempt to perform their best work under good conditions.
8. Requirements do not change.
9. The system is completely unclassified.
10. Hardware and software are in place and accessible at any time.
11. Some of the activities are not required but may affect productivity.
12. Design and code inspections occur without interruption and the time spent is 100 percent productive.
13. Time spent correcting errors found during the Software System Test phase is 100 percent productive.
14. At least one person must be assigned to an activity before it can begin accruing expenses.
15. Specifications do not change.
16. Modules have to go through design, code and test--in that order--before the total product can be successfully developed. However, portions of these phases can be scheduled simultaneously.
17. Product complexity remains constant.
18. Objectives of the project are already defined, and a feasibility analysis of the concept has been performed. The feasibility of the design remains for the user to determine.
19. The user is the project manager and performs no part in the technical detail. The project manager must rely on the expertise of the staff chosen.
20. Appropriate configuration management is performed by the project team. All generally accepted techniques for structured coding are followed.
21. Lines of code and pages of documentation measures for each person fall within appropriate ranges for average rates. The measures include documenting code as it is written. Employees are either documentation specialists or programmers. No employee writes code and documentation. Analysts write specifications, and programmers write code. (This is done to simplify the data base requirements and is not meant to conclude that programmers cannot write documentation.)
22. If too many people are assigned to an activity, productivity decreases. This also implements Brooks' philosophy of "adding people to a late software project makes it later." Productivity goes down and the length of time required to achieve the same level of productivity goes up.
23. Activities, as defined in the simulator, are independent.
24. When coding and reviews/inspections are scheduled in the same week, the coding is done first and the reviews/inspections follow.
25. The Waterfall life cycle is assumed to be the most efficient means of scheduling the tasks. However, the user can select tasks in any order, thus deviating from the Waterfall life cycle. Deviations are not as productive as the Waterfall life cycle.
26. Code is not reused.
27. Error correction does not create new errors.
28. Training and learning time are not required for the staff.

Figure 3: Assumptions of the Simulator

Total Source Instructions	16,500.00
Source Instructions Per Person-Month	238.40
Scheduled Months to Complete Requirements	2.10
Scheduled Months for Product Design	3.30
Scheduled Months for Code and Unit Test	5.10
Scheduled Months for Integration Test	2.90
Total Months to Complete	13.40
Person-Months of Effort in Requirements	4.80
Person-Months of Effort in Product Design	11.76
Person-Months of Effort in Detailed Design	18.00
Person-Months of Effort in Code and Unit Test	23.53
Person-Months of Effort in Integration Test	15.91
Total Person-Months of Effort	74.00

Figure 4: COCOMO Schedule and Effort Estimates

measures up to 10 percent more than the ones encompassed in Figure 4 are considered to be good.

The COCOMO estimates were compared to similar estimates from Gaffney's model and were found to be consistent. Using the equation

$$S = C * k^{.6288} * t^{.555} \text{ (Gaffney, 1983),}$$

the values for the effort in person-years (k) and development time in years (t) were calculated as shown in Table 10 for S = 16,500. For k = 5.76 and t = 0.94, C is calculated as approximately 5688, which is also consistent with Gaffney's model. While the COCOMO and Gaffney estimates are rough, initial estimates, they provide a range of reasonable values for the SPMS environment.

Table 10: Software Equation Estimates

C	k	t
3700	5.76	2.05
6100	5.76	.835
3700	11.37	.94
6100	5.14	.94

The life-cycle elements from which a user may choose consist of phases, activities, and tasks. The phases are System Definition, Software Design, Software Development, and Software Integration and System Testing, which must be performed in this order. Unit testing is performed as part of Software Development. Activities are major components of phases and consist of tasks. To keep the size of the life cycle and data base manageable, activities

are the smallest units of work to which programmers and analysts can be assigned. Tasks further describe activities and are provided as commentary in the simulator. The phases and their corresponding activities and tasks can be found in Appendix A. A user may assign start and end dates and the specific people who will perform the work. Not all activities of each phase must be completed before the next phase begins; there can be some concurrent activities. At some point in time near the end of any software development project, all phases have to be completed in the order stated above.

Intuitively, some analysis must be performed before designing a system, and a design must exist before coding can take place. The design may be nothing more than a programmer's or analyst's thoughts and ideas, but it is a design from which code can be written (Yourdon, 1989). Obviously, software must be coded before it can be tested. It is this reasoning that led to using these phases and allowing the user to select start and end dates for activities at will. What becomes important in successful execution of the SPMS is electing to perform activities in their proper order, where there is a proper order, and selecting certain productivity-improving activities within each phase. The ultimate goal is to select the right combination of activities, assign reasonable start and end dates to activities, determine the necessary quality improvement and testing procedures, and assign the appropriate people to each activity.

The flexibility of entering dates and people on activities provides the means to accommodate the incremental Waterfall life cycle discussed by Boehm in *Software Engineering Economics* (pp. 41-42) or any other life cycle. Once an activity is started, it remains open until the end date arrives. However, if there are no people assigned to an activity, there are no expenses directly charged to that activity, and no productive work is performed. Therefore, the user can leave an activity open with no people assigned and, in effect, stop the

work for that activity without terminating the activity. An activity is terminated when the end date assigned to it passes and terminated activities cannot be reopened.

To illustrate, if the user wants to simulate the Waterfall life cycle, where a software product is developed incrementally, each of the phases can be scheduled concurrently with people first assigned only to the System Definition phase. While the System Definition is being completed, people can be moved on to the Software Design phase activities and then on to Software Development once some design is completed. It is not necessary that all the requirements be found before some design begins or for a complete system design to be rendered before code is written. If a software system can be divided into independent work units, then the system can be incrementally developed. The simulation assumes this independence exists. However, if the user wishes to complete all the System Definition before the Software Design and other phases, then the development can be scheduled that way. The user completely defines the life cycle used by assigning the activity start and end dates.

Figure 5 is an example of a PERT chart that could result from an actual simulation. The user selected the start and end dates for each activity, resulting in the ordering of the elements shown in the chart. The Early Start Date was calculated automatically by the simulator and is the earliest possible time the activity may begin (the day after the latest End Date entered on predecessor activities).

The ideal model is also represented as a PERT chart of the same format, except that the dates are not completed. Each activity has a predecessor activity (except the beginning activity) and links to successor activities. As each week passes within the simulation, a list of the correct order of next activities is updated and compared against the user's selection of activities. Each activity also has an optimal duration in the ideal model, and this is compared

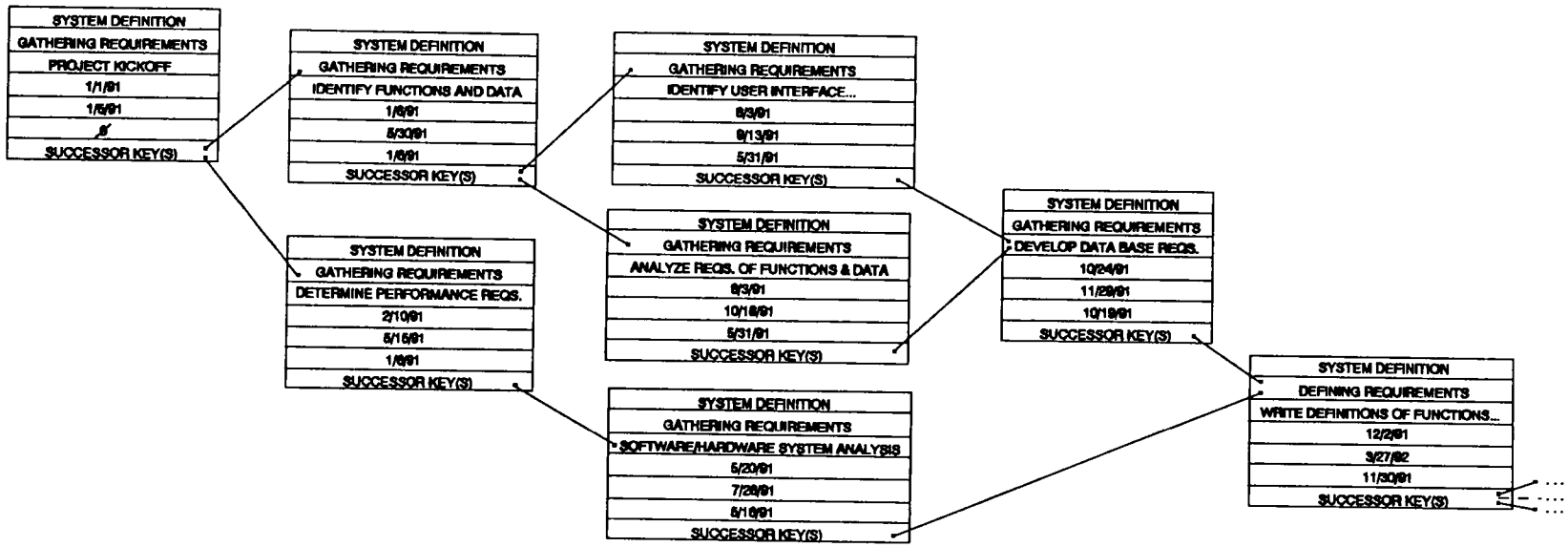


Figure 5: Example of PERT Chart

with the duration assigned by the user to determine progress. The algorithm that traverses the PERT chart to determine next correct activities is a modification of Knuth's topological sorting algorithm found in the first volume of *The Art of Computer Programming* (Knuth, 1973).

Linked to each activity of the ideal model are descriptions of what the user learns by scheduling activities in the proper order and assigning the appropriate staff to perform the work described. This description is revealed to the user weekly and may be reviewed at will. The system requirements are revealed in this manner as the user schedules client interviews to be conducted by the proper analysts. If interviews are scheduled for the proper duration, all the requirements will be learned by the user. If, however, the user does not schedule enough time for the interviews and ends the activity prematurely, requirements will be missing from the subsequent design and code. Thus the end product will not be exactly as the client wanted. This is also true for all the phases, so it is possible to have all or only a portion of any phase's output.

Designing the Simulator Data Base

The model developed in this thesis uses discrete data from models and observations in literature to identify the data elements required in the SPMS data base and to provide realistic boundaries for these elements and for productivity measures. In the SPMS, productivity is measured in lines of code and pages of documentation. Obviously, these units cannot be mixed within the same calculations. However, there is an associated cost for each unit, so dollars are always a common unit of measure for any activity. For those activities that

contribute to the completion of the software product but that do not produce source code, a relationship must be defined between the effort of these activities and their contribution to the total lines of code. Documentation is such an activity. For new projects it is difficult to determine how much documentation will be written and how it will contribute to writing code. Gathering product requirements is typically a documentation activity that directly affects the coding activity. For the purposes of the SPMS, project data reported by various authors was used to calculate an average number of lines of code resulting from a page of documentation. Obviously, the project environment and format of the documentation greatly affect these calculations. What is important to note is that a relationship does exist between the size of a final product in lines of code and the number of pages of associated documentation. A translation factor (lines of code per page of documentation) is maintained in the data base to provide a solution to this problem.

The goal of project completion is to assign resources to implement all functions (write the necessary number of lines of code) within the allotted budget and time constraints. So, the model has an optimal size, given in lines of code and pages of documentation, and an assigned budget. Each person working on the project is capable of writing a certain number of lines of code or documentation per hour, can remove a percentage of total errors during each phase, and has an associated hourly rate for performing these activities. There is also a limit to the number of hours a person can work in a week and still be productive. Elements are stored in the data base to allow the flexibility of varying the performance of individuals. A simplification of the idea behind the model is that overall productivity (LOC) is the sum of the optimal productivity of each person minus the effect of counterproductive factors in the development environment.

The data base elements required to measure the quality of the final product must also be considered. The product's quality is determined by the development process and the activities performed to detect errors. For the purposes of the SPMS, errors can be removed by assigning good programmers to coding activities, conducting design and code inspections, and planning and conducting thorough system testing. Discrete data base elements have been identified which allow these activities to be performed and measured.

The SPMS data base contains elements that pertain to activities, activity descriptions, employees (analysts and programmers), project productivity, expenses, reporting, and selecting activities correctly. The relationships between these elements are best described by Figure 6. A complete list of elements in each file structure shown in this figure are included in Figure 7.

For each user of the SPMS, a complete copy is made of the model to allow user-specific data to be maintained. Certain data in the Activity file is static, such as Phase ID, Activity ID, Activity Name, Optimal Duration, Required Lines of Code, and Area of Expertise. The values of other fields such as Start Date, End Date, and User's Name depend on what the user selects via the user-input screens.

Linked to the Activity file are files that maintain the user selections and give the user feedback concerning these selections. The Username file contains one record that stores the current simulation date and the parameters used in the productivity calculations. The testing and total project reports data records are linked to the Username file and contain feedback on the progress of the testing phase and the overall project. There is only one of each of these reports per user name, and each is updated weekly as the user signals the passage of weeks in the simulation.

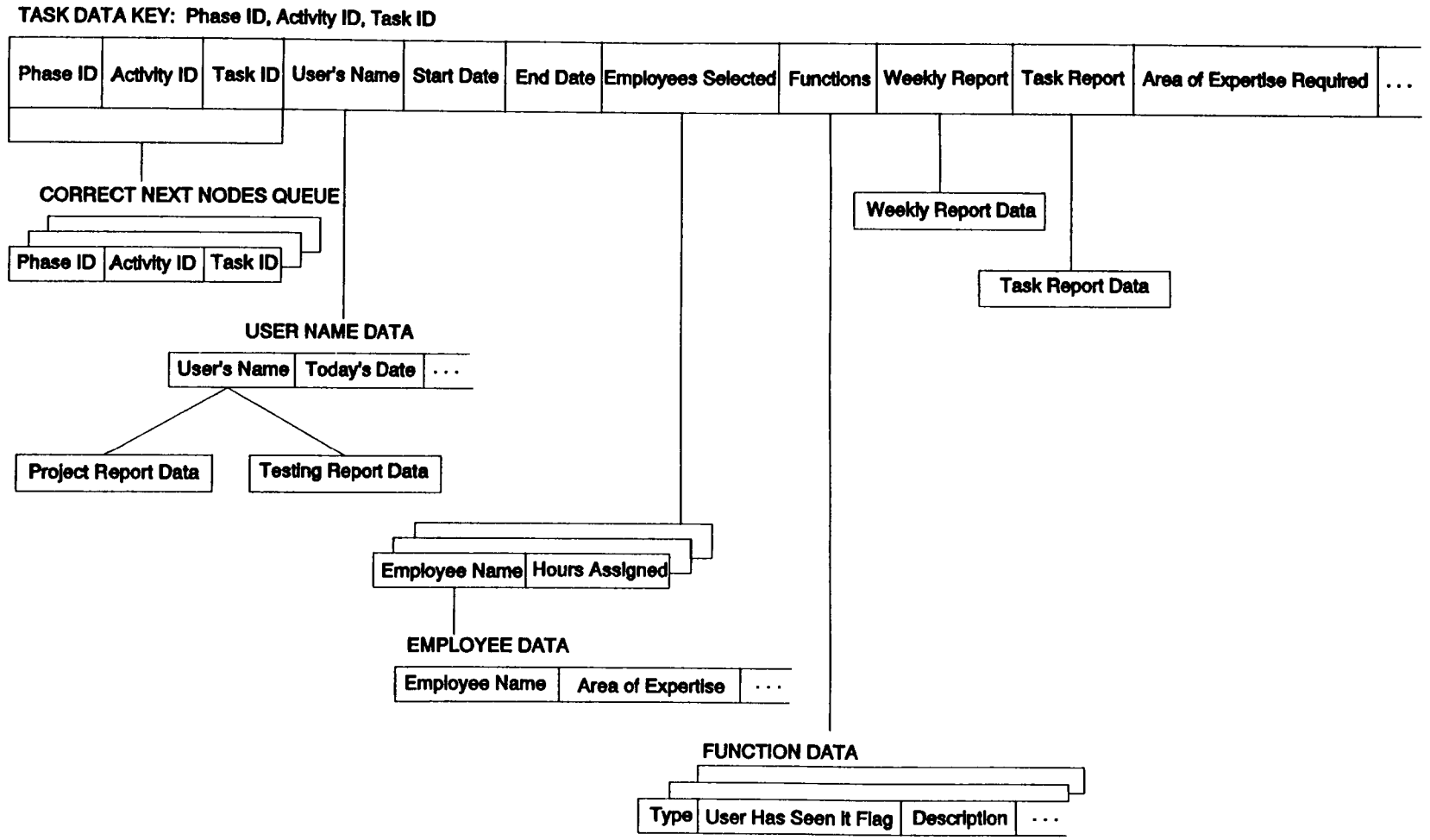


Figure 6: Relationship of Elements

Employee Data

Name
Title
Rate (dollars per hour)
Area of Expertise (programmer or analyst)
Lines of Code Written per Hour
Lines of Documentation Written per Hour
Number of Activities Assigned
Total Hours Assigned to Activity(s)
Definition Phase Error Removal Efficiency
Design Phase Error Removal Efficiency
Code and Unit Test Error Removal Efficiency
System and Integration Test Error Removal Efficiency
Resume

Activity Data

Phase ID
Activity ID
Activity Name
Start Date
End Date
Early Start Date
User's Name
Optimal Duration in Weeks
Optimal Manhours
Required Lines of Code to Complete
Required Pages of Documentation
Lines of Code Completed by the User
Pages of Documentation Completed by the User
Area of Expertise Required to Successfully Perform the Activity
Employees Selected (list)
Functions (list)
Report Data (pointer to Report Data records)
Error Removal Efficiency (for inspection and testing activities only)
User Has Selected This Activity
Number of Predecessors
List of Successors
Optional Flag
Selected Correctly

Weekly Report Data

Date of End of Report Week
Activity Name
Activity Start Date
Activity Target/Actual End Date
Activity Labor Hours
Activity Labor Cost
Activity Overhead
Activity Fee

Figure 7: Data Base Elements

Weekly Report Data (continued)

Activity Total Cost
Lines of Code Written on Activity
Pages of Documentation Written on Activity
Effective Working Hours
Effective Working Time Per Dollar
Maximum Effective Working Time Per Dollar

Activity Report Data

Activity Name
Activity Start Date
Activity Target/Actual End Date
Activity Labor Hours
Activity Labor Cost
Activity Overhead
Activity Fee
Activity Total Cost
Lines of Code Written on Activity
Pages of Documentation Written on Activity
Effective Working Hours
Effective Working Time Per Dollar
Maximum Effective Working Time Per Dollar

Project Report Data

Report Date
Total Project Labor Hours
Total Project Labor Cost
Total Project Overhead
Total Project Fee
Total Project Total Cost
Lines of Code Written on Total Project
Pages of Documentation Written on Total Project
Effective Working Hours
Effective Working Time per Dollar
Maximum Effective Working Time per Dollar

Username Data

User Name
Today's Date
Actual Weeks to Finish
Estimated Weeks to Finish
Total Lines of Code Developed
Total Pages of Documentation Developed
Lines of Code Per Page of Documentation
Lines of Code Not Reviewed in Definition Phase
Lines of Code Not Reviewed in Design Phase

Figure 7: (continued)

Username Data (continued)

Lines of Code Not Reviewed in Coding Phase
Lines of Code Not Tested
Total Errors Remaining in Definition Phase
Total Errors Remaining in Design Phase
Total Errors Remaining in Coding Phase
Total Errors Remaining in Test Phase
Errors per KLOC Attributed to Definition
Errors per KLOC Attributed to Design
Errors per KLOC Attributed to Coding
Effort to Correct an Error
Definition Error Correction Multiplier
Design Error Correction Multiplier
Coding Error Correction Multiplier
System Testing Error Correction Multiplier
Acceptance Testing Error Correction Multiplier
Lines of Code Tested Per Person-Hour
Lines of Code Reviewed Per Person-Hour
Maximum Work Hours Per Week
Definition Phase Review Error Removal Efficiency
Design Phase Review Error Removal Efficiency
Coding Phase Review Error Removal Efficiency
Link to Project Expenses Report
Link to Testing Report

Function Data

Function Type
User Has Seen This Feedback
User Feedback

Correct Next Nodes Queue

List of Activities for the Next Nodes in the Correct System Model
Estimated Weeks to Finish
Total Lines of Code Developed
Total Pages of Documentation Developed
Lines of Code Per Page of Documentation
Lines of Code Not Reviewed in Definition Phase
Lines of Code Not Reviewed in Design Phase

Figure 7: (continued)

Also linked to the Activity file is a list of the employees assigned by the user to each activity. The user enters the number of hours per week that each employee is to devote to this activity. Associated with each activity is the proper area of expertise (programmer or analyst) for the employees working on the activity. In order for work to be productive, an employee's area of expertise must match the area of expertise required by the activity. The work to be performed on each activity is described as either programming or documentation. Programmers perform the coding activities and analysts perform the documentation activities. While this is not always the case in reality, this assumption greatly simplifies the SPMS model.

The Function Data file contains the descriptions of the functions of the fictional client's organization and the progress of the project team. These descriptions are revealed to the user when an activity is properly scheduled and progress is made toward its completion.

The weekly and activity report data records are updated for each week that an activity is active (the start date has passed and the end date has not yet arrived). The user may view these reports at any time via a pull-down menu. Layouts for all the reports are shown in Appendix B.

The information in the Correct Next Nodes Queue is created by the SPMS and is used to determine the next correct activity(s) that should be selected by the user in order to show progress. This data structure is a queue that contains the key to each of the currently active activity's successors; it is used to traverse the PERT chart and determine whether the user has scheduled activities properly.

Designing the Control Flow and Algorithms

The data files and structures shown in Figure 7 contain the minimum number of elements to provide the following SPMS functions:

- providing help and activity descriptions to guide the user,
- scheduling activities,
- assigning programmers and analysts to activities,
- assigning weekly hours for each person on an activity,
- determining productivity measures per person (lines of code and pages of documentation),
- calculating error removal efficiencies by phase and person,
- determining the cost and effort of error correction,
- conducting design and code inspections,
- determining the correct order of activity selections,
- judging the user's performance,
- testing the final software product and producing testing reports, and
- providing feedback via weekly, activity, and project expense and productivity reports.

These functions are implemented in a menu-driven system with pull-down windows that guide the user through the development of the software product. The overall flow of control through the SPMS is depicted in Figure 8 using the Box Description Language (Mills, 1986). The user decides when to exit the simulator by selecting the "Quit" option in the pull-down menu. Until that time the user may select actions to perform from a Weekly Action List. This option leads to assigning people to perform the activities. The user may schedule as

```
[Clear data base fields and control variables.  First time thru = Yes]
Use BB Initialize Variables
Use BB Display Introductory Screens
[Have the user enter a name to use in creating a personal copy of the model.  This allows the user
to exit and save the simulation data and return to continue execution later.]
Use BB Prompt for User's Name
If Fatal Error = No then
        Use BB Show Project Phases
        Use BB Get Today's Date
        Date a Week Ago = Today's Date
        [Repeat until the user selects "Yes-Leave Simulator" in the pull-down menu.  A user can do
        this at any time.]
        Repeat While User Selection NE "Yes-Leave Simulator"
                If first time thru = No then
                        Use BB Add One Week To Today's Date
                        Use BB Increment Actual Weeks to Finish
                        Use BB Validate Activity Network
                        Use BB Validate Optional Activities
                        Use BB Show New Report Data is Available
                        Date a Week Ago = Today's Date
                fi
                Use BB Show Weekly Action List
                First Time Thru = No
        Loop
        Use BB Confirm Leaving and Saving
else
        Use BB Display Fatal Error and Quit
fi
```

Figure 8: SPMS Control Flow

many activities as desired. When the user returns to the Weekly Action List screen, a week of time passes and messages are displayed indicating that progress has been made toward completing the final product. The management reports are updated at this time, and the user may select an option in the pull-down menu to review the new reports. Appendix C provides additional detail and screen layouts which demonstrate the complete program.

Each of the previously listed functions deals with determining the productivity of individuals, tracking project progress, and measuring the quality of the final product. These concepts were implemented by combining information from various sources. Esterling's model (1980) was used to determine the effective working time of individuals and considers the number of people assigned to a single activity in the calculation. Therefore, the more people assigned to an activity, the lower the productivity for each individual. Tracking project progress requires linking expenses to each activity, determining the size of the final product, and determining the contribution of each activity toward the final product, as well as considering the rate at which documentation and code are written. The quality of the final product is measured by the number of errors that remain after testing is completed. A number of errors is assumed to be inherent in the design and code. Error removal activities must be performed to detect and correct these errors in each phase of development.

Determining the productivity of individuals requires calculating the number of hours spent productively programming or writing documentation. This involves reviewing the conditions under which the work is performed and considering the effects of communicating with others working on the same activity, the number of interruptions, and the time spent on administrative activities. Also, if a person is asked to work overtime to complete an activity, it is necessary to know the maximum number of hours that a person can work and still produce a good product. The Esterling model (1980) is used to determine the fraction of a

day spent productively programming for each individual. This fraction is then multiplied by the maximum lines of code that each person can produce to determine the total productivity per day per person. Summing these values for every programmer gives the total project productivity in lines of code. Lines of documentation are counted in the same manner and then multiplied by the conversion factor "Lines of Code per Page of Documentation." Figure 9 contains an overview of the processing performed in the module to determine the productivity of each individual.

As an SPMS user schedules activities and assigns people to perform the required work, the progress toward completing the final product is tracked and revealed to the user. Each activity in the ideal model has a predetermined number of associated lines of code or pages of documentation. In order to complete each activity, the people assigned to the activity must collectively produce the required number of lines of code or pages of documentation. The user does not know the ideal size associated with each activity, but is shown the results of the work performed via management reports (see Appendix B) and is given enough information through help screens and activity descriptions to estimate the size of the activities from the total project size. The effective working hours are shown in each of the reports to provide feedback on the effectiveness of the team assigned to each activity. This figure is used to calculate the effective working time per dollar that is a part of each report.

The total cost of the development effort is also tracked in the management reports by activity and by total project. The cost is determined by the number of hours and hourly rates assigned to each person. The hourly rate is part of each employee's resume and is accessible to the SPMS user. Labor cost for each activity is simply calculated by multiplying the hours for each employee assigned to an activity by the corresponding hourly rate and

Administrative Fraction = .1
Regain Train of Thought = 2
Number of Interruptions per Day = 4
Interruption Duration = 5
Number of Project-Related Interruptions = 3

If this activity has been selected by the user and is active then
Activity Weekly Effective Hours = 0
Activity Weekly LOC = 0
Activity Weekly POD = 0

For each employee assigned to this activity

Employee Weekly LOC = 0

Employee Weekly POD = 0

If the Area of Expertise for the employee = Area of Expertise required by the activity then

Number of Hours Overtime on This Activity = 0

[Assume an employee cannot be assigned more than Maximum Hours per Week, including all activities to which they are assigned.]

If the Total Hours Assigned to Employee > Maximum Hours per Week then

Possible Hours Overtime = Maximum Hours per Week - 40

Percentage of Overtime Hours on This Activity = Number Hours Assigned on This Activity / Total Hours Assigned to Employee

Number of Hours Overtime on This Activity = Possible Hours Overtime * Percentage of Overtime Hours on This Activity

fi

Fraction Effective = (40 + Number of Hours Overtime on This Activity - 40 * Administrative Fraction - 20 * Regain Train of Thought/60 - 5 * Number of Interruptions per Day * (Interruption Duration + Regain-Thought)/60 - 5 * Number of Project-Related Interruptions * (Number of People on Activity - 1) * (Interruption Duration + Regain Train of Thought)/60)/40

Number of Effective Hours = Number of Hours Assigned on This Activity * Fraction Effective

If this is a coding activity then

Employee Weekly LOC = LOC per Hour for Employee * Number of Effective Hours

Figure 9: Determine Employees' Productivity

Activity Weekly LOC = Activity Weekly LOC + Employee Weekly LOC

fi

If this is a documentation activity then

Employee Weekly POD = POD per Hour for Employee * Number of Effective Hours

Activity Weekly POD = Activity Weekly POD + Employee Weekly POD

Employee Weekly LOC = Employee Weekly POD * Lines of Code per Page of Documentation

Activity Weekly LOC = Activity Weekly LOC + Employee Weekly LOC

fi

fi

od

fi

Figure 9: (continued)

summing these figures for all the employees assigned. The total project labor cost is the sum of the activity labor costs. Also included in the labor cost are overhead and contract fee rates. In the current SPMS, overhead and fee rates are set at 80 percent and 15 percent, respectively, of the labor cost. The overhead rate reflects the cost of doing business and maintaining support staff. The contract fee is negotiated with the client. These figures assume that the software development is taking place in a large company and that the user has access to any resources required to complete the project.

The quality of the software product completed in the SPMS is determined by the number of errors that remain in the product after system testing has ended. System and integration testing is performed automatically when the user schedules this phase. Various authors suggest that there are errors inherent in software which will remain in the product if not found by error removal techniques, such as design and code inspections and testing. The number of errors in the product developed in the simulation are calculated based on this observation. In the literature reviewed for the simulator, error rates for untested code were found to range from 30 to 85 errors per thousand lines of code.

The simulation begins with the number of total errors set to a maximum. These errors are divided among the Definition, Design, and Coding phases according to the percentage of errors likely to be attributed to each phase to obtain an error density for each phase. From the Software Error Introduction and Removal Model found in Boehm (1981), 5 percent of the total errors in a product can be attributed to definition, 25 percent to design, 15 percent to code, and 15 percent to documentation. This gives an overall error rate of 60 errors per KLOC. For the purposes of this simulation, only the errors attributed to the Definition, Design, and Coding phases are considered. To simplify the SPMS, documentation is assumed to be a by-product of the development process and is performed in each of the

phases. Therefore, the error rate assigned to documentation is equally divided among the development phases. Another major assumption of the SPMS is that errors do not occur during the system testing phase. Here again, this assumption simplifies the prototype model.

Each phase shown in the SPMS contains error removal activities with associated error removal efficiencies. The number of errors removed by each error removal activity is subtracted from the total errors for the corresponding phase. For example, if there are 2000 lines of code attributed to the Coding phase and 25 errors per KLOC is the error density for the Coding phase, then one could expect 50 errors to occur in the Coding phase. If code inspections that remove 10 percent of coding errors are conducted, then 5 errors would be removed, leaving 45 total Coding errors. Each of the values involved in calculating the number of errors remaining in the product is an element in the data base and can be set in order to change the simulation. Error removal efficiencies of several activities were given in Chapter 2. The error removal activities that are included in the SPMS are:

- requirements review,
- design inspection,
- code inspection,
- unit testing, and
- system testing.

Unit testing is automatically performed during the coding phase and its efficiency may be different for each person assigned to the coding activities. The other error removal activities must be scheduled by the user.

The effort and cost to correct errors are also considered in the model and are phase-sensitive so that the sooner errors are found, the less expensive they are to fix. To demonstrate this fact, a single value is used as the effort to correct any error, which is multiplied by the corresponding phase multiplier. The effort and the phase multipliers are

distinct elements in the data base. The corresponding values can be set to affect different scenarios. In *Software Metrics* (Grady and Caswell, 1987), the number of hours required to correct errors found in code was found to range from 2 to 50 hours, if the errors were found and corrected during the testing phase. Correcting documentation requires even less time per error. The SPMS does not consider documentation errors. It was found that there is insufficient empirical data reported in literature to adequately define an environment and identify an accurate value for the effort required to make corrections in each life-cycle phase. However, a data base element exists to accommodate a value for the effort to correct an error so that its effects can be studied.

From a graph reported by Boehm (1981, p. 40) showing the relative cost to correct an error in each phase of software development, the following phase multipliers were derived:

Requirements	2.5
Design	5.0
Code	10
Development Testing	18
Acceptance Testing	50

If the effort to correct an error is 5 hours, the hourly rate of the person assigned to the current testing activity is $50.00, and the error is found during the coding phase, then the cost of correcting the error is given by 5 * 50 * 10, or $2,500. The above multipliers may not be correct for any specific environment, but represent one possible scenario in the simulation.

In this version of the simulator there is no choice of testing methods. The only requirement of testing is to have written code in the selection of previous activities. The duration of the testing phase is the number of weeks the user schedules. Each week a test report prepared in the format shown in Figure 10 is displayed, showing the results of testing. The number of errors remaining in the code is determined by subtracting the number of

	For the Past Week	Total Project
System Definition	999,999	999,999
Software Design	999,999	999,999
Software Development	999,999	999,999
Total	9,999,999	9,999,999
Lines of Code Tested	999,999	999,999
Cost of Error Removal	$9,999,999	$9,999,999

Figure 10: Total Errors Found and Corrected by Phase

errors removed by performing the review and testing activities from the total number of errors in the product before testing.

Capers Jones (1986) provides error removal efficiencies for the review and testing activities that are applied in the simulation. The combined efficiency of all error removal methods will not be greater than 99 percent, as suggested by Jones. The user must determine when to stop testing by reviewing the test report data. If the user schedules activities properly, requirements will be learned and the product will be designed and coded. If error removal activities are also performed, then the product will contain fewer errors than if these activities had not been performed. The data in the test reports will then reveal fewer errors in many lines of code. If the user did not schedule activities properly, the product will not be completely coded and will contain many errors. The error reports would then show many errors in the very few lines of code tested each week. This assumes, of course, that the user schedules an adequate amount of time for testing. If testing is not performed for the proper duration, then errors will remain in the final product and it may fail the client's acceptance test.

In order to properly interpret the simple data in the test reports, the SPMS user must have an understanding of the complexities of the testing process. As is the case in the real world, the user will not know the error density of the final product and will not know when all errors have been removed from the product. The decision to stop testing should consider cost, schedule, and the observed error detection rates.

CHAPTER 4. OBSERVATIONS FROM USING THE SPMS

The COCOMO estimates given previously show a schedule of 13.4 months and 74 person-months of effort to write 16,500 lines of code. The total project cost is estimated as $319,800 by the COCOMO model. Using the Esterling model parameters set for "typical programmers" and holding them constant while varying the average number of people assigned to the project, the data in Table 11 are derived. A 40-hour workweek is assumed. This is only an approximation of one scenario's results. If the user schedules fewer than six people over the total development time, the schedule will not be met. If the total project staffing averages more than seven, development cost will exceed the budget. Assuming the average cost per month for each person is $4,000, then the development cost for seven people is $319,200 and $345,600 for eight people. As the number of people increases, the total project development time decreases and cost increases beyond the budget. When 10 people are required to interact, the productivity begins to decline until reaching zero at 20 people. At this point, the cost increases to $80,000 per month with no productivity. These calculations use only the Esterling model.

Using the model described by Gaffney (1983), the values in Table 11 can be further studied to determine a reasonable schedule. Gaffney suggests using the equation $S = C * k^p * t^q$ to determine the value of the development constant, C, from a given schedule and proposed effort.

If the value of C is within the range for a specific environment, then the schedule and effort may be reasonable. If the value of C is too high, then the schedule may be too short;

Table 11: Schedule and Effort Estimates

People	Fraction Effective	LOC/PM	Effective Hours	Schedule (Months)	PM of Effort
10	.43	158	688	10.4	104
9	.48	177	691	10.4	93
8	.52	191	666	10.8	86
7	.56	206	627	11.4	80
6	.61	224	586	12.3	74
5	.65	239	520	13.8	69
4	.69	254	442	16.2	65
3	.73	269	350	20.5	61
2	.78	287	250	28.7	57
1	.83	305	133	54.0	54

if the value of C is too low, then the schedule may be longer than necessary. Using the data in Table 11, the value of C was calculated as shown in Table 12. The range for C that was developed for the IBM Federal Systems Division is 3700 to 6100. Notice in Table 12 that all values for C are within this range except one. Therefore, the schedules calculated for 2 to 10 people are probably reasonable. Staffing decisions should consider the schedule that satisfies the goal of 13.4 months and $319,800. Within the SPMS, the optimal average number of people to assign throughout the development of the sample project is six.

The SPMS model and data are derived from the models described by Boehm (1981), Gaffney (1983), Putnam (1978), and Esterling (1980). Boehm's COCOMO model is used to provide effort and schedule estimates for the sample project. The Esterling model is used to determine the effective working time per week for each person assigned to the project. This value is used to determine the productivity of each person. Limitations on the optimum number of people to assign to the project and their levels of productivity are imposed from the results of applying the models of Gaffney and Putnam in the manner described to create the data in Table 11 and Table 12.

Table 12: Schedule, Effort, and the Development Constant

People	Schedule (Months)	PM of Effort	Development Constant (C)
10	10.4	104	4,549
9	10.4	93	4,928
8	10.8	86	5,015
7	11.4	80	5,155
6	12.3	74	5,322
5	13.8	69	5,000
4	16.2	65	4,741
3	20.5	61	4,533
2	28.7	57	3,819
1	54.0	54	2,759

Before any lines of code or pages of documentation are written, the user must assign activities in the order given by the underlying PERT chart. The user is not shown the order in which the activities must be assigned. When an activity is scheduled correctly, documentation or code is written by each person assigned to the activity at the rate each is given in the data base. To complete the project on time, average productivity should be close to 224 LOC/PM as shown in Table 11. Any lower productivity will require the user to extend the schedule to complete the project.

After attaining some measurable productivity, the user may schedule design and/or code inspections to locate and correct errors. The user must determine the duration of the inspections without prior knowledge of the exact rate at which code can be inspected. After a week of inspections, reports reveal the number of lines of documentation and code reviewed during that week so the user can calculate the inspection rate. The user decides when to stop conducting inspections. If inspections are stopped before all code is reviewed, there will be errors left in the product. These errors may or may not be found in the testing phase. Also, the later that errors are corrected, the more costly the corrections. The user

is shown the cost of corrections performed as a result of inspections and testing. Errors corrected during the testing phase will cost 13 to 30 times those found during inspections. The exact difference in cost depends on which value is assigned in the underlying model. The user will see the difference via inspection and testing reports. However, the user will not know the cost of correcting an error during system testing until that phase is scheduled. (For a complete example of one execution of the SPMS, see Appendix C.) Because the underlying model and variable settings are not revealed, the user learns by making good and bad decisions and seeing the outcome in the simulation. Sometimes the outcome will not be apparent until very late in the project. If early decisions lead to later problems, the increased cost becomes apparent. In subsequent executions of the simulator, the user will have the opportunity to develop the hypothetical software again and provide different decisions, thus effecting a different outcome. In the prototype SPMS, the lessons a user may learn consist of:

- lines of code are not a sufficient measure of productivity,
- the productivity among programmers may vary as much as 10 to 1,
- removing errors early is less costly than removing the same errors later,
- spending time gathering requirements and creating a good design saves time in coding and testing,
- once a software development project is behind schedule, the only way to meet the schedule is to eliminate functionality,
- determining factors that most affect productivity is difficult in an unfamiliar environment,
- developing a reasonable project schedule demands an understanding of all factors that can affect productivity,
- measuring the productivity of individuals in an attempt to use the information for schedule and effort estimation is difficult,

- consequences of a project manager's decisions throughout the life of a software project are not always known until the project ends,
- the cause/effect relationships that result from a project manager's decisions are difficult to identify,
- it is possible to develop a complete software product that is not what the client wanted, and
- the quality of a software product is only as good as the quality of the process that created it.

CHAPTER 5. CONCLUSIONS

Most software effort and cost models involve estimating the size of the final product in lines of code. This can be a highly error-prone process, especially if there is no historical data on which to base an estimate. All sources referenced here stress the importance of collecting local data in order to improve the estimation techniques within a specific environment. The SPMS allows the use of a base of data collected from an existing environment to provide realistic ranges for lines of code estimates, personnel productivity, and schedule. It is recognized that the data drawn from literature in order to form this knowledge base represents many different environments, therefore it is used primarily to demonstrate the principles of the simulation and is not intended to predict the outcome of a project or duplicate a specific project's results.

The prototype SPMS provides for only one correct software development process and life cycle. However, the algorithms and data that describe the prototype SPMS's model provide a framework from which one can build a complete model that can be used to aid decision-making and project management training. The parameters in the model can be set to correspond to an existing software development environment, thus providing a means to study the effects of various productivity factors on a schedule and budget within that environment.

Using the SPMS or similar model requires collecting data that is specific to the environment to be studied. It is also important to understand the data that is collected and its proper use in measuring productivity and software quality. For lack of a better means to measure productivity, lines of code are used as a measure. There are many pitfalls associated

with using lines of code written per programmer-hour and lines of code inspected or tested per person-hour.

Measuring the quality of software must also be approached carefully. It has been reported by many authors that design and code inspections are effective in removing errors in software. However, the quality of the inspection process depends on the experience of the individuals conducting the reviews. The quality of the testing process also depends on the expertise of the testers. Software developers must realize that "quality cannot be tested into a program" (Jones, 1978).

The SPMS is intended to aid the study of the software development process and the many factors that affect the quality of software. In order to make definitive statements concerning the many possible cause/effect relationships among the data and observations that exist in the software development environment, more convincing evidence is needed to validate these data. As more data are collected across a variety of computer applications, perhaps stronger correlations between the many aspects of the software development process and the quality of the resulting software product can be found. To facilitate the study of such data and processes as presented here, more software tools are needed to lead the computer industry toward defining a more scientific and less error-prone approach to developing software. Perhaps someday quality software can be a guaranteed by-product of the software development process.

BIBLIOGRAPHY

BIBLIOGRAPHY

Abdel-Hamid, T.K. "The Dynamics of Software Project Staffing: A System Dynamics Based Simulation Approach." *IEEE Transactions on Software Engineering*, February 1989, Vol. 15, No. 2, pp. 109-119.

Boehm, Barry. *Software Engineering Economics*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1981.

Brooks, Frederick P. *The Mythical Man-Month: Essays on Software Engineering.* Reading, Massachusetts: Addison-Wesley Publishing Company, 1975.

Esterling, R. "Software Manpower Costs: A Model." *Datamation*, March 1980, pp. 164-170.

Fagan, M.E. "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal*, 1976, Vol. 15, No. 3, pp. 182-211.

Gaffney, John E., Jr. "Approaches to Estimating and Controlling Software Costs." Paper presented at the CMG XIV International Conference on Computer Performance Evaluation, Crystal City, Virginia, December 1983.

Grady, Robert B., and Deborah L. Caswell. *Software Metrics*. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1987.

Horowitz, Ellis, and Sartaj Sahni. *Fundamentals of Data Structures*. Potomac, Maryland: Computer Science Press, Inc., 1976.

Humphrey, Watts S. *Managing the Software Process*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1989.

Jones, Capers. *Programming Productivity*. New York: McGraw-Hill Book Company, 1986.

Jones, T.C. "Measuring Programming Quality and Productivity." *IBM Systems Journal*, Vol. 17, No. 1, 1978, pp. 39-63.

Knuth, Donald E. *The Art of Computer Programming, Vol. 1, Fundamental Algorithms.* Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1973, pp. 251-265.

Martin, Charles F. *User-Centered Requirements Analysis*. Englewood Cliffs, New Jersey: Prentice Hall, 1988.

Martin, James, and Steven Oxman. *Building Expert Systems: A Tutorial.* Englewood Cliffs, New Jersey: Prentice-Hall, 1988.

Mills, Harlan D., Richard C. Linger, and Alan R. Hevner. *Principles of Information Systems Analysis and Design*, Orlando, Florida: Academic Press, Inc., 1986.

Myers, G.J. "A Controlled Experiment in Program Testing and Code Walkthroughs/ Inspections." *Communications of the ACM*, Vol. 21(9), September 1978, pp. 760-768.

Putnam, L.H. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem." *IEEE Transactions on Software Engineering*, July 1978, Vol. SE4, No. 4, pp. 345-361.

Quinnan, R.E. "The Management of Software Engineering, Part V: Software Engineering Management Practices." *IBM Systems Journal*, 1980, Vol. 19, No. 4, pp. 466-477.

Walston, C.E., and C.P. Felix. "A Method of Programming Measurement and Estimation." *IBM Systems Journal*, 1977, Vol. 16, No. 1, pp. 54-73.

Yourdon, Edward, *Modern Structured Analysis*. Englewood Cliffs, New Jersey: Yourdon Press, 1989.

APPENDICES

APPENDIX A: LIFE-CYCLE PHASES AND ACTIVITIES

APPENDIX A: LIFE-CYCLE PHASES AND ACTIVITIES

Phase: System Definition (Requirements Analysis)

Activity: Initial Project Meeting with Client Management

Tasks
Meet with Client Management
Become Familiar with Client's Expectations for Performance

Activity: Identify Functions and Data of the Customer Organization

Tasks
Interview Users
Interview Department Managers
Identify Business Functions that are Candidates for Automation
Identify Data/Entities/Objects Used
Identify Business Functional Relationships (Processes)
Identify Data Relationships and Interactions - role of data in org/connections
Collect Existing Documents (manuals, forms, etc.)
Determine Company Standards, Policies, and Procedures
Document System Functions as Identified
Document Data Relationships

Activity: Analyze Requirements of Functions and Data Used

Tasks
Group Data According to Functions (Subjects)
Establish Services (identify services)
Determine Attributes of Data/Entities/Objects and Instance Connections
Identify Current and Future Manual Processes
Identify Current and Future Automated Processes
Determine Interfaces Between Manual and Automated Processes
Determine Interfaces with the "Outside World" and Other Systems
Identify Data Partitions or Classes
Establish System Constraints and Limitations
Compile Information into Software System Requirements Specifications
Response Time to Inputs
Reliability Requirements and Fail Safe Measures
Turn-around Time for Reports and Data Update
Environmental Constraints - type of equip. that can be used in machine shop etc.

Activity: Software/Hardware System Analysis

Tasks
Determine What Hardware Can be Used
Analyze Current and Future Memory Utilization
Review Computer Operations - How does data get to hardware? Where and how is data stored?
Determine Maintenance Requirements

Activity: Identify User Interface Requirements

Tasks
Screens - input/output formats
Reports - input/output formats - batch and online
External System Behavior - method of data entry, batch and online processing

Activity: Develop Data Base Requirements

Tasks
Quantity of Data to Store
Historical Tracking of Data
Data Entry Methods
Error Correction
Data Update Frequency
Analyze System Model to Determine What & How Data is Stored
Access Methods Study

Activity: Write Definitions of Functions and Data (diagrams, math model)

Tasks
Structured English Specifications - template, define services
Data Flow Diagrams
Entity/Object Relationship Diagrams
State Transition Diagrams
Structure Charts
Context Diagram
Events List
State-Event-Response Table
Integrate Diagrams and Text Into a Model (one document)
Check Model Consistency
Check Model Redundancy
Using System Model, Create User Dictionary
Create Definitions of Data/Entity/Object Stores
Compile Information into Software System Description Document

Activity: Produce Deliverables

Tasks
Send Documentation to Editing and Proofreading
Coordinate Graphics Production for Documentation
Reproduce Final Documentation
Review Draft Documentation
Review Final Documentation
Client Review of Draft Material
Management Review
Quality Assurance Reviews

Activity: Review of Requirements

Tasks
Formal Group Inspection
Perform Changes Resulting from Inspections (Iterate on Change & Review of Change)
Internal Management Review
Formal Proof of Correctness
Review Software System Requirements Specifications
Validate Requirements Specifications
Team Review and Formal Validation

Activity: Review Requirements with Client

Provide Client a Copy of the Deliverables
Meet with Client to Review Deliverables
Request Client Approval of Requirements
Meet with Client to Deliver Final Requirements Specifications
Get Client Approval to Continue Project

Activity: Develop Software Development Plan

Tasks
Establish Implementation Strategy (top-down, bottom-up...)
Write Testing Plan
Determine Configuration Management Plan
Identify Hardware and Software Tools Required for Development
Documentation Plans - Outlines
Document Preparation and Editing

Phase: Software Design

Activity: Internal Review of Software Design

Tasks
Formal Module Design Inspection
Hardware/Software Architecture Review
Review Integration Plan
Review Test Plan and Script
Group Technical Review of Final Design Specifications
Review Specifications Documentation Before Delivery to Client
Review Deliverables

Activity: Subsystem and Program Design

Tasks
Analyze Functions and Data Collected During Definition Phase
Analyze Control Flow
Map Functions and Data to Subsystems
Determine Subsystems from Requirements
Decompose the Subsystems into Components
Develop Preliminary Structure Charts
Breakdown Components into Physical Programs
Map External View (User View) to Physical Programs
Determine Program Inputs and Outputs (Interface Specifications)
Define Transitions (Transformation of Inputs and Outputs)
Create System Specifications Document

Activity: Develop Detailed Program Specifications

Tasks
Construct Diagrams of Data Usage and Flow Within the Physical System
Enhance Preliminary Structure Charts
Develop Detailed Program Design (Select Algorithms)
Detail Design Using a Design Description Language
Compile All Specifications in a Project Notebook
Update System Specifications Document

Activity: Review Design with Client

Tasks
Provide Client a Copy of Deliverables
Match Requirements to Design (Determine Completeness of Design)
Get Client Approval to Continue Development Effort

Activity: Test Design

Tasks
Determine Test Cases
Write Testing Script from Requirements and Specifications
Create Test Data Files

Activity: Develop Test Plan

Tasks
Determine Testing Methodology - depends on life cycle chosen
Conduct a Technical Review of Test Plan
Determine Source of Test Data (Client and/or Derived)

Activity: Produce Deliverables

Tasks
Edit and Prepare System Specifications Documentation
Edit and Prepare Test Plan Documentation
Edit and Prepare Integration Plan

Activity: Develop Integration Plan

Tasks
Review Specifications Document and Study Program Interfaces
Review the Test Plan
Coordinate System Integration with Test Plan
Write and Edit Integration Plan

Activity: Hardware Design

Tasks
Determine Hardware and Software Interaction Requirements
Determine Memory Requirements and Existing Memory Availability
Compare System Access Requirements to Actual Access Time
Perform Benchmarking Tests to Estimate Actual Response Time
Determine the Quantity of Data to be Manipulated and Response Times
Select Appropriate Hardware

Phase: Software Development

Activity: Module Development

Tasks
Review Coding Standards and Structured Coding Techniques
Code Control Modules
Code User Interface
Code Reports
Code Complex Algorithms and Computation Modules
Write Brief Program Documentation as Necessary
Update Pseudocode as Corrections are Necessary
Maintain Project Notebook via Librarian
Maintain Program Libraries via Configuration Management

Activity: Software Testing

Tasks
Unit Test of Modules Performed by Programmers
Problem Analysis and Error Correction (Back Up and Fix Design)
Conduct Preliminary Function Testing of Related Modules if Required

Activity: Internal Software Review

Tasks
Informal Peer Code Reviews
Verification of Code Against Design
Formal Program Code Inspections

Activity: Integrate Software Modules

Tasks
Review Integration Plan
Perform Top-Down Integration of Programs

Phase: Software System Test

Activity: Perform Integration Testing

Tasks
Verify Programs Interface Correctly
Report and Document Errors as Found
Correct Errors as Found

Activity: Perform System Testing

Tasks
Execute Test Script
Report and Document Errors as Found
Correct Errors as Found

APPENDIX B: SPMS SCREEN AND REPORT LAYOUTS

APPENDIX B: SPMS SCREEN AND REPORT LAYOUTS

Welcome Screen

Welcome to Software Perfection, Inc.

You are an employee of Software Perfection, Inc. (SPI). You will manage a project to define, design, develop, and test software for a decision support system (DSS) that will be used by your client, Johnson Manufacturing, Inc. (JMI) to collect data concerning their manufacturing process and the quality of their products. JMI believes that the DSS will lead to better products thus giving them an edge over their competition.

If the DSS is delivered on time and within budget, JMI will pay a $30,000 bonus to SPI (of which you get a cut). If the software is delivered late, or the project exceeds the budget, SPI must forfeit the bonus. If the project exceeds the budget, you and SPI are responsible for the amount of the overrun. In other words, no matter what the cost, a software product has to be delivered and accepted by JMI.

You have 14 months to deliver a complete DSS to JMI. It is estimated that the DSS will require approximately 16,500 lines of code and $320,000 for labor costs only. Company overhead and fee are additional expenses you must consider. The total budget is estimated to be $625,000.

Press Enter to continue....

Enter your 8-character project team name:

_____

A first-time user will automatically go through all subsequent screens. Users who have previously run the simulator are prompted with "Review the project management and planning screens? (Y/N): _" (See next screen layout.) A previous simulation can be resumed by entering the proper project team name.

The user enters a name and presses Enter. The name must not correspond to a system file name and cannot be blank. If the name entered is invalid, the user is prompted to reenter the name.

If this is a new user, while creating the user's files, display "Creating files. Please wait...." . The user does not have to press a key to continue. The next screen is displayed as soon as the files are created.

Enter your 8-character project team name:

_____

Review the project management and planning screens? (Y/N): __

If the user enters "Y" or "y" then display all subsequent screens.
If the user enters "N" or "n" then go to Weekly Action List screen.

If the user enters any other character, display "Invalid input - enter Y or N."

Project Phases

Your primary contact is John Smith. All client interaction must be coordinated with him. The philosophy at SPI is to create the best product for the lowest cost. It has been shown that following a methodology helps to achieve this goal. Your methodology will include:

System Definition
Software Design
Software Development
Software System Test
Acceptance Test

It is up to you to decide when each phase begins and ends.

Press Enter to continue....

Project Activities Screen

Project Activities

Each phase consists of the following activities.

1. System Definition
   - Initial Project Meeting with Client
   - Identify Functions and Data
   - Software/Hardware System Analysis
   - Analyze Reqs of Functions and Data
   - Identify User Interface Reqs
   - Define Functions and Data
   - Develop Data Base Requirements
   - Produce Deliverables
   - Develop Software Development Plan
   - Review of Requirements
   - Review Requirements with Client

2. Software Design
   - Hardware Design
   - Subsystem and Program Design
   - Develop Detailed Program Specs
   - Develop Integration Plan
   - Develop Test Plan
   - Test Design
   - Produce Deliverables
   - Review Design with Client
   - Internal Review of Software Design

3. Software Development
   - Module Development
   - Integrate Software Modules
   - Internal Software Review

4. Software System Test
   - Perform Integration Testing
   - Perform System Testing

5. Acceptance Test
   - Perform Client Acceptance Test

Press Enter to continue....

Planning Work

In the following screens you will be required to make decisions pertaining to the schedule and performance of the previously listed activities. The following information will have to be entered or selected from lists in order to perform each activity:

Start Date
End Date
Employees Assigned
Employee Hours

Performing Work

After scheduling activities and assigning employees, you should select the "Perform the Work" option from the following screen to advance the date and have the work be performed by the employees you assigned.

Press Enter to continue....

Evaluating Work

Reports that aid evaluation of your decisions are available each week via the pull-down menu in subsequent screens. Definitions of terms you should know to understand the reports are given below.

Labor Hours:	The total hours employees are scheduled to work.
Labor Cost:	Labor Hours * Employee Rate
Overhead:	Administrative costs of SPI. 80% of Labor Cost.
Fee:	The contract fee negotiated between SPI and JMI that allows SPI to make a profit. A fee of 15% of Labor Cost is added to the total cost billed to JMI.
Potential Effective Working Hours: (PEWH)	The total hours scheduled for employees to work, up to 60 hours per week, minus the hours spent on administrative duties, various interruptions and meetings. It is assumed that all employees have duties beyond analysis and coding.
Actual Effective Working Hours:	PEWH minus the effects of improper sequencing and staffing of activities on the critical path to project completion (i.e., project management errors).

Press Enter to continue....

Evaluating Work (page 2)

Lines of Code (LOC):	The primary measure of productivity. Every coding activity directly generates LOC according to the coding rates of the individuals assigned to the activity.
Pages of Documentation (POD):	Includes all requirements and design specs, user and operator manuals, and program documentation. In order to use a common unit of measure for productivity, POD are converted to LOC using a conversion factor, LOC per POD.
Percent of Budget Spent:	Total Cost / Total Budget, where Total Cost is Labor Cost + Overhead + Fee and Total Budget is $625,000.
Percent of Schedule Spent:	Months Passed in Simulation / Total Months (14)
Percent of Product Completed:	LOC Completed / Total LOC (16,500)

Press Enter to continue....

Review Help Quit

Weekly Action List

Today's Date: mo/da/yr

Action List

1. Plan the Work

2. Perform the Work

Select an action (1-2): __

F10 (Pull-down Menu)

If the user selects option 1, go to Select a Phase screen.
If the user selects option 2, display Enter Number of Weeks to Pass Window.

The pull-down menu may be activated by pressing the F10 key in any screen where the pull-down menu is displayed. The pull-down menu contains the following options. An option is selected by using the arrow keys to move up, down, left, and right and pressing the Enter key. The ESC key is used to exit an option in the pull-down and to close the pull-down menu to return to the Action List screen.

Review	Help	Quit
Resumes	About This Screen...	Yes - leave simulator
Reports	Definitions	No - do not leave simulator
Project Feedback	Activities	

Enter Number of Weeks to Pass Window

Today's Date: mo/da/yr

Enter the number of weeks you wish to advance Today's Date or zero to exit without advancing the date:

_0

Note: Processing will continue until all weeks have passed.

The number of weeks entered by the user must be greater than or equal to 0 and less than or equal to 99. If the number of weeks entered is less than 0 or greater than 99, display on line 23 "Invalid entry - weeks must be between 0 and 99, inclusive."

After the user enters a valid number of weeks, display centered on line 10 "Processing... please wait".

For active activities (i.e., start date is less than Today's Date and end date is greater than Today's Date), determine if any Project Feedback screens need to be displayed. If any to display, display each screen of feedback, allowing the user to press Enter to continue to the next feedback screen. After all feedback screens have been displayed, continue processing. One Project Feedback screen is displayed for each week an activity is active.

Determine if design or code inspection activities are active and display the Software Design Review screen and/or the Software Review screen as appropriate.

When the processing is finished, display centered "Updated reports are available. Press any key to continue." on line 10. When the user presses a key, return to the Weekly Action List screen.

Time passes according to the number of weeks entered by the user regardless of the state of the simulator. Processing cannot be interrupted while the time is passing.

Software Design Review Weekly Report

Pages of Design Documentation Reviewed: 999,999

Errors Removed From Design: 999,999

Note: Software Design Review consists of design inspections performed according to the procedures outlined by M.E. Fagan in the paper "Design and code inspections to reduce errors in program development" published in the IBM Systems Journal, Vol. 15., 1976.

Press Enter to continue....

Software Review Weekly Report

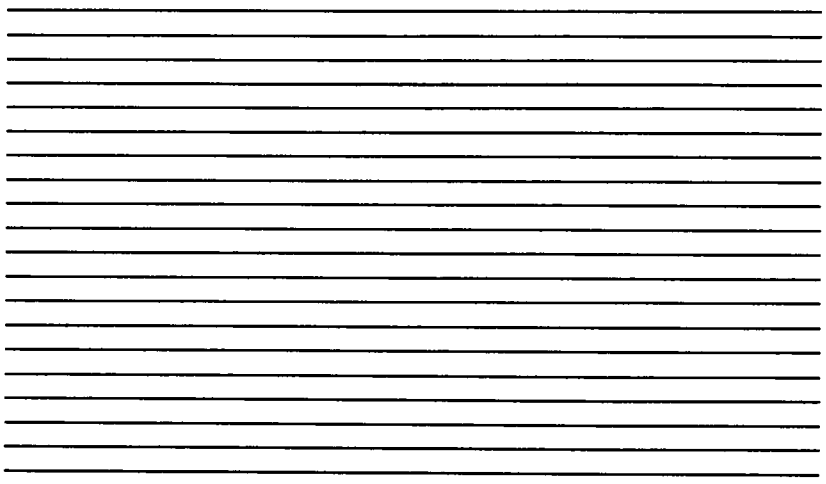
Lines of Code Reviewed: 999,999

Errors Removed From Code: 999,999

Note: Software Review consists of code inspections performed according to the procedures outlined by M.E. Fagan in the paper "Design and code inspections to reduce errors in program development" published in the IBM Systems Journal, Vol. 15., 1976.

Press Enter to continue....

Feedback for Activity: _____



Press Enter to continue....

This screen contains the feedback to the user for each active activity. One screen of feedback is displayed for each week an activity is active.

Review Help Quit

Select a Phase

Today's Date: mo/da/yr

Life Cycle Phases

1. System Definition
2. Software Design
3. Software Development
4. Software System Test
5. Acceptance Test

Select a phase (1-5): _

F2 (Finished), F10 (Pull-down Menu)

If the user presses F2, ignore any data entered in the input field and return to the Weekly Action List screen.

Review Help Quit

Select an Activity

Today's Date: mo/da/yr

{Window Title}
Activities List

1. (activity 1)
2. (activity 2)
3. (activity 3)
4. (activity 4)
5. (activity 5)
6. (activity 6)
7. (activity 7)
8. (activity 8)
9. (activity 9)
10. (activity 10)
11. (activity 10)

Select an activity (1 - {end}): __

F2 (Finished), F10 (Pull-down Menu)

{Window Title} is replaced with the name of the phase the user selected.

The activities in the Activities List are read from the data base and filled in according to the activities found for the phase selected by the user. Activities cannot be deleted or added in this screen, only chosen by the user.

{end} is replaced with the number of activities in the list.

Eleven is the maximum number of activities per phase.

If the user presses F2, any data entered in the input field is ignored and the user is returned to the Select a Phase screen.

Assign Dates

Review Help Quit

Today's Date: mo/da/yr

{Window Title}

Start Date (MMDDYY): _____

End Date (MMDDYY): _____

F1 (Exit Without Change), F2 (Finished), F3 (Assign Employees)
F10 (Pull-down Menu)

The user can assign dates without assigning employees and assign employees without assigning dates.

{Window Title} is the Activity Name of the activity chosen by the user.

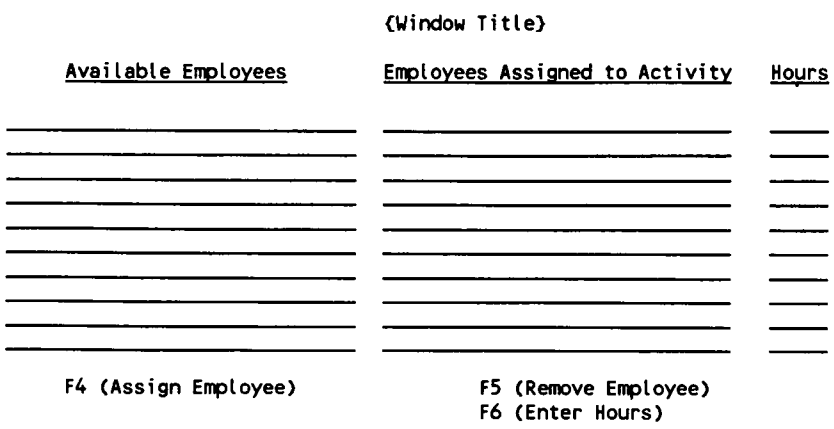
The user may enter zeros for both dates if the activity has not yet started. The Enter key is used to move from one date field to another.

If the user presses F1, any data in the input fields is ignored and the user is returned to the Select an Activity screen.

If the user presses F2, the dates entered are validated. If no error has occurred, return to the Select an Activity screen. If there is an error, display the appropriate message and allow the user to correct the error. Possible error messages are:

"This date does not exist - reenter"
"Caution! Either both dates must be entered or neither date entered."
"Caution! End Date must be later than or the same as Start Date."
"Caution! Cannot retroactively start an activity."
"Warning! This activity has already started. You can't change the start date."
"Warning! You can't retroactively end an activity."
"This activity has already ended. You cannot change it!"
"Start Date must be greater than or equal to Today's Date!"

If the user presses F3, ignore any data entered and go to the Assign Employees screen.



The user can only select employees in either list. Names cannot be typed over. An employee is selected by using the arrow keys to position the light bar on an employee name and pressing a function key.

If F1 (Exit Without Change) is selected, any changes the user has made are ignored and control returns to the Assign Dates screen.

If F2 (Finished) is selected, update the data base with changes made by the user and return to the Assign Dates screen.

If F3 (See Resume) is selected, display the resume for the employee whose name is highlighted.

If F4 (Assign Employee) is selected, a name in the Available Employees list must be highlighted. If no name is highlighted in the Available Employees list and this list is not empty, display on line 23 "Highlight a name in the Available Employees list. Press Enter to continue." If a name is highlighted in the Available Employees list, move the highlighted employee name to the first blank line in the Employees Assigned to Activity list, erase the name selected from the Available Employees list and move remaining names in the Available Employees list to fill the blank line created. The light bar remains on the employee just assigned. To enter hours for the employee, the user must press F6.

If F5 (Remove Employee) is selected, the user must have highlighted an employee in the Employees Assigned list. If no name in the Employees Assigned list was highlighted and the list is not empty, display "Highlight a name in the Employees Assigned list. Press Enter to continue." If a name in the Employees Assigned list is selected, this employee name is moved to the next empty line in the Available Employees list, the corresponding hours are erased, the name is erased from the Employees Assigned list and remaining names are moved to fill the blank created. The user may remove all employees from the Employees Assigned to Activity list for any activity.

If F6 (Enter Hours) is selected, the user must have highlighted a name in the Employees Assigned list. If a name was highlighted, disable F1, F2, F3, F4, F5, and F6 until the user enters valid hours or ESC to exit without changing the hours. If the light bar is not on a name in the Employees Assigned list, display "Highlight a name in the Employees Assigned list. Press Enter to continue." If the light bar is on a name in the Employees Assigned list, the user can enter hours for the corresponding employee. The user presses Enter when they have completed entering hours or ESC to exit the hours field without changing the previous value. The hours entered must be between zero and 60, inclusive. If there is an error, display on line 23 "Hours must be zero or <= 60". Press Enter to continue or ESC to cancel." and position the cursor on the leftmost position of the hours field in error. If there is no error, the hours in the Employees Assigned to Activity list are changed for the assigned employee (the data base is not updated yet).

See Resume

Resume of {Employee Name}

Title _____ Rate (per hour) $999.99

Education

_____

Experience

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Press Enter to continue....

The information on this screen is read only, the user cannot update it here. When the user presses Enter, return to the Assign Employees screen.

This screen format is also used in the Review Resumes option of the pull-down menu.

Activity Weekly Report

Expenses and Productivity for the Week Ending 99/99/99 [11]

Activity: [activity name]

Start Date: MM/DD/YY End Date: MM/DD/YY

Labor Hours: [1]

Actual Effective Working Hours: [8]
Potential Effective Working Hours: [14]

Labor Cost: [2]
Overhead: [3]
Fee: [4]

Total Cost: 9,999,999 [5]

Effective Working Time per Labor Dollar: [9]
Potential Effective Working Time per Labor Dollar: [10]

Lines of Code: [6]
Pages of Doc.: [7]

Cost per LOC: [12]
Cost per POD: [13]

Press Enter to continue....

Calculation

[1] The total hours for the past week for this activity entered by the user in the Assign Employees screen.

[2] Each assigned employee's rate multiplied by the hours assigned by the user.

[3] [2] * .80

[4] [2] * .15

[5] [2] + [3] + [4]

[6] The rate at which assigned employees write code multiplied by [8].

[7] The rate at which assigned employees write a page of documentation multiplied by [8].

[8] Potential Effective Working Hours [14] minus the effects of improper sequencing and staffing of activities on the critical path to project completion (i.e., project management errors).

[9] [8] / [2]

[10] [14] / [2]

[11] Today's Date at the time the report is created.

[12] [5] / [6]

[13] [5] / [7]

[14] The total hours scheduled for employees to work, up to 60 hours per week, minus the hours spent on administrative duties, various interruptions and meetings. It is assumed that all employees have duties beyond analysis and coding.

Activity Cumulative Report

Total Expenses for Activity

Activity: [activity name]

Start Date: MM/DD/YY End Date: MM/DD/YY

Labor Hours: [1]

Actual Effective Working Hours: [8]
Potential Effective Working Hours: [13]

Labor Cost: [2]
Overhead: [3]
Fee: [4]

Total Cost: 9,999,999 [5]

Effective Working Time per Labor Dollar: [9]
Potential Effective Working Time per Labor Dollar: [10]

Lines of Code: [6]
Pages of Doc.: [7]

Cost per LOC: [11]
Cost per POD: [12]

Press Enter to continue....

Calculation

[1] The total hours for this activity entered by the user in the Assign Employees screen.

[2] Each assigned employee's rate multiplied by the hours assigned by the user.

[3] [2] * .80

[4] [2] * .15

[5] [2] + [3] + [4]

[6] The rate at which assigned employees write code multiplied by [8].

[7] The rate at which assigned employees write a page of documentation multiplied by [8].

[8] Potential Effective Working Hours [13] minus the effects of improper sequencing and staffing of activities on the critical path to project completion (i.e., project management errors).

[9] [8] / [2]

[10] [13] / [2]

[11] [5] / [6]

[12] [5] / [7]

[13] The total hours scheduled for employees to work, up to 60 hours per week, minus the hours spent on administrative duties, various interruptions and meetings. It is assumed that all employees have duties beyond analysis and coding.

Total Project Expenses

Report Date: [1]

Labor Hours: [2]

Actual Effective Working Hours: [9]
Potential Effective Working Hours: [17]

Labor Cost: [3]
Overhead: [4]
Fee: [5]

Total Cost: 9,999,999 [6]

Effective Working Time per Labor Dollar: [10]
Potential Effective Working Time per Labor Dollar: [11]

Lines of Code: [7]
Pages of Doc.: [8]

Cost per LOC: [12]
Cost per POD: [13]

Percent of Budget Spent: [14]
Percent of Schedule Spent: [15]

Percent of Product Completed: [16]

Press Enter to continue....

Calculation

[1] Today's Date at the time the report is generated.

[2] The total hours for the project entered by the user in the Assign Employees screen.

[3] Each assigned employee's rate multiplied by the hours assigned by the user.

[4] [3] * .80

[5] [3] * .15

[6] [3] + [4] + [5]

[7] The rate at which assigned employees write code multiplied by [9].

[8] The rate at which assigned employees write a page of documentation multiplied by [9].

[9] Potential Effective Working Hours [17] minus the effects of improper sequencing and staffing of activities on the critical path to project completion (i.e., project management errors).

[10] [9] / [3]

[11] [17] / [3]

[12] [6] / [7]

[13] [6] / [8]

[14] [6] / Budget Assigned ($625,000)

[15] Actual Weeks Finished by the User / Estimated Weeks to Finish (56)

[16] LOC Completed by the User / Total Required LOC (16,500)

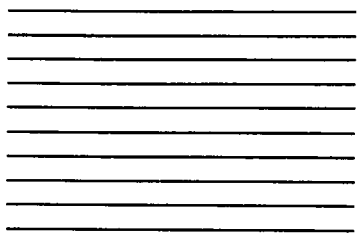
[17] The total hours scheduled for employees to work, up to 60 hours per week, minus the hours spent on administrative duties, various interruptions and meetings. It is assumed that all employees have duties beyond analysis and coding.

Total Errors Found and Corrected by Phase

	For the Past Week	Total Project
System Definition	999,999	999,999
Software Design	999,999	999,999
Software Development	999,999	999,999
Total	9,999,999	9,999,999
Lines of Code Tested	999,999	999,999
Cost of Error Removal	9,999,999	9,999,999

Press Enter to continue....

Review Resumes



Employee names appear in the window. Use the up and down arrow keys to move the light bar to a name and press Enter to view the resume for the employee selected. See the See Resume screen for the format of the resume to display. While the resume is displayed, press Enter to return to this window and select another name or exit. Press ESC to roll up the pull-down window one level.

Reports

Activity Report
Project Report
Testing Report

This is a window in the pull-down menu.

If the user selects Activity Report, display the Pull-down Activity Selection screen. When the user exits that screen they will return to this menu for further selection.

If the user selects Project Report, display the Project Report screen. When the user exits the project report screen by pressing Enter, they return to this menu for further selection.

If the user selects Testing Report, display the Testing Report screen. After the user views the report and presses Enter, return to this window for further selection.

Project Activities

System Definition
1._____
2._____
3._____
4._____
5._____
6._____
7._____
8._____
9._____
10._____
11._____

Software Development
21._____
22._____
23._____

Software Design
12._____
13._____
14._____
15._____
16._____
17._____
18._____
19._____
20._____

Software System Test
24._____
25._____

Acceptance Test
26.Perform Client Acceptance Test

Select an activity (1-27): ___

F2 (Finished)

The user may select an activity that has a title filled in.

Report Selection Usage

After entering an activity number and pressing Enter, the user is prompted with "Select Weekly or Cumulative Report (W or C): __" If the user selects W, display the Activity Weekly Report screen. If the user selects C, display the Cumulative Activity Report screen. After viewing either report, the user presses Enter to return to this screen for additional selection.

If the user presses F2, exit this screen and return to the Pull-down Report Menu Window.

Project Feedback Usage

This selection screen is also used to review the feedback the user was shown while advancing Today's Date in the Perform Work option of the Action List screen. After selecting an activity that has a title filled in, find the project feedback for the activity, and display the information in the format shown in the Project Feedback layout. When the user exits the feedback review screen, they return to this menu for further selection.

If the user presses F2, exit this screen and return to the pull-down menu.

Client Acceptance Test Results

The product delivered to the client consists of {x} lines of code.

Product functionality was found to be {complete/incomplete}.

The results of acceptance testing shows a possible {y} errors per KLOC remaining in the product which is {acceptable/unacceptable}.

Based on the information above, the client has {accepted/refused} the product.
{Congratulations./Sorry, try again.}

This screen is shown at the end of the simulation when the user has selected the Perform Client Acceptance Test activity. The text in brackets {} is replaced with the appropriate diagnostic message.

APPENDIX C: SPMS SAMPLE EXECUTION

APPENDIX C: SPMS SAMPLE EXECUTION

The sample execution of the SPMS included in this appendix represents one scenario in which every activity is performed with the proper employees and is scheduled correctly. The schedule that was entered via the simulator's input screens is shown in Table 13. The relevant assumptions of this simulation are:

- every employees' rate is $25.00 per hour,
- the lines of code written per hour are the same for all employees,
- the pages of documentation written per hour are the same for all employees,
- the error removal efficiencies are the same for all employees,
- all employees work 40-hour weeks,
- the error rate at the start of the simulation is 50 errors per KLOC,
- errors removed during the Software System Test phase require 25 times more effort than removing the same errors during the Software Design phase,
- Software Design Review error removal efficiency is 75%, and
- Software Review error removal efficiency is 70%.

Actual screen output from this simulation follows. Only feedback and reports for selected activities are shown. Examples of all user-input screens are included, as well as the introductory screens. Online help screens are not included. For more information on actual simulator control flow and calculations see the Prototype Software Project Management Simulator Design document.

Notice in the Total Project Expenses report at the end of the simulation that the required lines of code and pages of documentation are written and the product is delivered

Table 13: Schedule for a Sample Execution

Phase / Activity	Start Date	End Date	Number of Employees	Hours per Week
System Definition				
Initial Project Meeting	05/13/91	05/13/91	1	4
Identify Functions and Data	05/14/91	06/07/91	3	120
Software/Hardware System	05/14/91	05/26/91	1	40
Analyze Reqs of Functions	06/10/91	06/28/91	2	80
Identify User Interface Reqs	06/10/91	06/21/91	1	40
Define Functions and Data	06/10/91	07/05/91	1	40
Develop Data Base Reqs	06/24/91	07/05/91	1	40
Produce Deliverables	07/08/91	07/12/91	1	40
Develop Software Develop. Plan	06/24/91	07/05/91	1	40
Review of Requirements	07/01/91	07/05/91	3	120
Review Reqs with Client	07/15/91	07/19/91	1	40
Software Design				
Hardware Design	07/22/91	08/09/91	2	80
Subsystem and Program Design	07/22/91	08/09/91	3	120
Develop Detailed Program Specs	08/12/91	10/04/91	3	120
Develop Integration Plan	08/12/91	10/04/91	1	40
Develop Test Plan	08/12/91	10/04/91	1	40
Test Design	08/12/91	10/04/91	1	40
Produce Deliverables	10/07/91	10/18/91	1	40
Review Design with Client	10/21/91	10/25/91	1	40
Internal Rev of Software Design	10/07/91	10/18/91	3	120
Software Development				
Module Development	10/21/91	02/21/92	4	160
Integrate Software Modules	02/24/92	03/13/92	4	160
Internal Software Review	02/24/92	03/13/92	4	160
Software System Test				
Perform Integration Testing	03/16/92	04/10/92	4	160
Perform System Testing	04/13/92	06/12/92	4	160
Acceptance Test				
Perform Client Acceptance Test	06/15/92	06/19/92	1	40

on time, however, the development effort exceeds the budget. Several factors contribute to the overrun including setting the cost of error removal during system testing to 25 times the cost of error removal during the Software Design phase. Additional time spent during design and code review activities would reduce the number of errors remaining in the product before system testing, thus lowering the project cost. Also, the error rate of 50 KLOC could be lowered so that there are fewer errors to find in the product. Another contributing factor is that the models used to estimate the project effort and schedule do not explicitly consider error removal efficiencies or rates of reviewing documentation and code.

Welcome to Software Perfection, Inc.

You are an employee of Software Perfection, Inc. (SPI). You will manage a project to define, design, develop, and test software for a decision support system (DSS) that will be used by your client, Johnson Manufacturing, Inc. (JMI) to collect data concerning their manufacturing process and the quality of their products. JMI believes that the DSS will lead to better products, thus giving them an edge over their competition.

If the DSS is delivered on time and within budget, JMI will pay a $30,000 bonus to SPI (of which you get a cut). If the software is delivered late, or the project exceeds the budget, SPI must forfeit the bonus. If the project exceeds the budget, you and SPI are responsible for the amount of the overrun. In other words, no matter what the cost, a software product has to be delivered and accepted by JMI.

You have 14 months to deliver a complete DSS to JMI. It is estimated that the DSS will require approximately 16,500 lines of code and $320,000 for labor costs only. Company overhead and fee are additional expenses you must consider. The total budget is estimated to be $625,000.

Press Enter to continue....

Enter your 8-character project team name:

Project Phases

Your primary contact is John Smith. All client interaction must be coordinated with him. The philosophy at SPI is to create the best product for the lowest cost. It has been shown that following a methodology helps to achieve this goal. Your methodology will include:

System Definition
Software Design
Software Development
Software System Test
Acceptance Test

It is up to you to decide when each phase begins and ends.

Press Enter to continue....

Project Activities

Each phase consists of the following activities.

1. System Definition
   - Initial Project Meeting with Client
   - Identify Functions and Data
   - Software/Hardware System Analysis
   - Analyze Reqs of Functions and Data
   - Identify User Interface Requirements
   - Define Functions and Data
   - Develop Data Base Requirements
   - Produce Deliverables
   - Develop Software Development Plan
   - Review of Requirements
   - Review Requirements with Client

2. Software Design
   - Hardware Design
   - Subsystem and Program Design
   - Develop Detailed Program Specs
   - Develop Integration Plan
   - Develop Test Plan
   - Test Design
   - Produce Deliverables
   - Review Design with Client
   - Internal Review of Software Design

3. Software Development
   - Module Development
   - Integrate Software Modules
   - Internal Software Review

4. Software System Test
   - Perform Integration Testing
   - Perform System Testing

5. Acceptance Test
   - Perform Client Acceptance Test

Press Enter to continue....

Planning Work

In the following screens you will be required to make decisions pertaining to the schedule and performance of the previously listed activities. The following information will have to be entered or selected from lists in order to perform each activity:

Start Date
End Date
Employees Assigned
Employee Hours

Performing Work

After scheduling activities and assigning employees, you should select the "Perform the Work" option from the following screen to advance the date and have the work be performed by the employees you assigned.

Press Enter to continue....

Evaluating Work

Reports that aid evaluation of your decisions are available each week via the pull-down menu in subsequent screens. Definitions of terms you should know to understand the reports are given below.

Labor Hours:	The total hours employees are scheduled to work.
Labor Cost:	Labor Hours * Employee Rate
Overhead:	Administrative costs of SPI. 80% of Labor Cost.
Fee:	The contract fee negotiated between SPI and JMI that allows SPI to make a profit. A fee of 15% of Labor Cost is added to the total cost billed to JMI.
Potential Effective Working Hours: (PEWH)	The total hours scheduled for employees to work, up to 60 hours per week, minus the hours spent on administrative duties, various interruptions and meetings. It is assumed that all employees have duties beyond analysis and coding.
Actual Effective Working Hours:	PEWH minus the effects of improper sequencing and staffing of activities on the critical path to project completion (i.e., project management errors).

Press Enter to continue....

Evaluating Work (page 2)

Lines of Code (LOC):	The primary measure of productivity. Every coding activity directly generates LOC according to the coding rates of the individuals assigned to the activity.
Pages of Documentation (POD):	Includes all requirements and design specs, user and operator manuals, and program documentation. In order to use a common unit of measure for productivity, POD are converted to LOC using a conversion factor, LOC per POD.
Percent of Budget Spent:	Total Cost / Total Budget, where Total Cost is Labor Cost + Overhead + Fee and Total Budget is $625,000.
Percent of Schedule Spent:	Months Passed in Simulation / Total Months (14)
Percent of Product Completed:	LOC Completed / Total LOC (16,500)

Press Enter to continue....

Review Help Quit

Today's Date: 05/20/91

Action List

1. Plan Work
2. Perform Work

Select an action (1-2): 1

F10 (Pull-Down Menu)

Review Help Quit

Today's Date: 05/20/91

Life Cycle Phases

1. Software Definition
2. Software Design
3. Software Development
4. Software System Test
5. Acceptance Test

Select a phase (1-5): 1

F2 (Finished), F10 (Pull-down Menu)

Review Help Quit

Today's Date: 05/20/91

Software Definition
Activities List

1. Initial Project Meeting with Client
2. Identify Functions and Data
3. Software/Hardware System Analysis
4. Analyze Reqs of Functions and Data
5. Identify User Interface Reqs
6. Define Functions and Data
7. Develop Data Base Requirements
8. Produce Deliverables
9. Develop Software Development Plan
10. Review of Requirements
11. Review Requirements with Client

Select an activity (1-11): 1

F2 (Finished), F10 (Pull-down Menu)

Review Help Quit

Today's Date: 05/20/91

Initial Project Meeting with Client

Start Date (MM/DD/YY): 05/13/91

End Date (MM/DD/YY): 05/13/91

F1 (Exit Without Change), F2 (Finished), F3 (Assign Employees)
F10 (Pull-down Menu)

Initial Project Meeting with Client

Available Employees	Employees Assigned to Activity	Hours
David Adams	Marvin Harris	4
Harriet Mario		
Jane Bellows		
John Dowling		
Julie Turner		
Mary Stevens		
Richard Donaldson		
Samuel Sams		
Susan Cleary		

F4 (Assign Employee) F5 (Remove Employee)
F6 (Enter Hours)

F1 (Exit Without Change), F2 (Finished), F3 (See Resume)

Resume of Marvin Harris

Title: System Analyst Rate (per hour):$ 25.00

Education
Master in Computer Science

Experience
Mr. Harris has 10 years experience as a programmer of information systems and 10 years experience in information system analysis and design.

Press Enter to Continue...

Review Help Quit

Today's Date: 05/20/91

Action List

1. Plan Work
2. Perform Work

Select an action (1-2): 2

F10 (Pull-Down Menu)

Enter the number of weeks you wish to advance Today's Date or
zero to exit without advancing the date:

5

Note: Processing will continue until all weeks have passed.

Initial Project Meeting with Client Management

The client wants a decision support system to aid in managing the scheduling of manufacturing operations, ordering raw materials, and determining cost effective measures for improving the manufacturing process.

Management will provide any documentation concerning their current systems and processing and would like for your project team to interview department managers and users to determine their needs for a decision support system. The focus of the development should be on providing information concerning the scheduling of the manufacturing process and the timely ordering of raw materials.

Press Enter to continue....

Identify Functions and Data of the Client Organization

The client's department managers and users were interviewed this week. Documentation such as manuals for existing automated systems, forms and processing instructions, and standards, policies, and procedures manuals were collected.

The department managers are familiar with structured design methodologies and advocate their use. The client is very knowledgeable of the type of information that you need in order to determine system requirements and is cooperating fully. Most of the users are computer literate and are very cooperative.

Press Enter to continue....

Software/Hardware System Analysis

These tasks began this week:

- o Determine what existing hardware can be used.
- o Analyze current and future memory utilization.
- o Review computer operations - How does data get to the hardware? Where and how is data stored?
- o Determine maintenance requirements.

The project team will finish reviewing the client's existing computer operations this week.

Press Enter to continue....

Identify Functions and Data of the Client Organization

These tasks continued this week:

- o Identify business functions that are candidates for automation.
- o Identify data/entities/objects used in the client's business.
- o Identify business functional relationships (processes).
- o Identify data relationships and interactions -
  the role of the data in the client's organization.

These tasks began this week:

- o Document system functions as identified.
- o Document data relationships.

The project team has been documenting functions and data relationships as the information became available, so they believe they will finish the documentation this week.

Press Enter to continue....

Identify Functions and Data of the Client Organization

These tasks began this week:

- o Identify business functions that are candidates for automation.
- o Identify data/entities/objects used in the client's business.
- o Identify business functional relationships (processes).
- o Identify data relationships and interactions -
  the role of the data in the client's organization.

The project team has identified approximately half of the business functions and corresponding data. They anticipate working one more week on these tasks.

Press Enter to continue....

Software/Hardware System Analysis

The project team finished reviewing the client's computer operations and worked on the following tasks:

- o Determine what existing hardware can be used.
- o Analyze current and future memory utilization.
- o Determine maintenance requirements.

Press Enter to continue....

Identify Functions and Data of the Client Organization

The following tasks continued this week:

- o Interview users.
- o Interview department managers.
- o Collect existing documents (manuals, forms, etc.)
- o Determine company standards, policies, and procedures.

Press Enter to continue....

Updated reports are available. Press Enter to continue.

Expenses and Productivity for the Week Ending 05/20/91

Activity: Initial Project Meeting with Client

Start Date: 05/13/91 End Date: 05/13/91

Labor Hours:	4	Actual Effective Working Hours:	3
		Potential Effective Working Hours:	3
Labor Cost:	100	Effective Working Time per Labor Dollar:	0.03
Overhead:	80	Potential Effective Working Time per Labor Dollar:	0.03
Fee:	15		
Total Cost:	195		

Lines of Code:	0	Cost per LOC:	0
Pages of Doc.:	0	Cost per POD:	0

Press Enter to continue...

Expenses and Productivity for the Week Ending 06/10/91

Activity: Identify Functions and Data

Start Date: 05/14/91 End Date: 06/07/91

Labor Hours:	120	Actual Effective Working Hours:	89
		Potential Effective Working Hours:	89
Labor Cost:	3000	Effective Working Time per Labor Dollar:	0.03
Overhead:	2400	Potential Effective Working Time	
Fee:	450	per Labor Dollar:	0.03

Total Cost:	5,850		

Lines of Code:	279	Cost per LOC:	21
Pages of Doc.:	8	Cost per POD:	731

Press Enter to continue...

Total Expenses for Activity

Activity: Identify Functions and Data

Start Date: 05/14/91 End Date: 06/07/91

Labor Hours:	480	Actual Effective Working Hours:	374
		Potential Effective Working Hours:	374
Labor Cost:	12000	Effective Working Time per Labor Dollar:	0.03
Overhead:	9600	Potential Effective Working Time	
Fee:	1800	per Labor Dollar:	0.03

Total Cost:	23,400		

Lines of Code:	1173	Cost per LOC:	20
Pages of Doc.:	34	Cost per POD:	688

Press Enter to continue...

Expenses and Productivity for the Week Ending 05/27/91

Activity: Software/Hardware System Analysis

Start Date: 05/14/91 End Date: 05/26/91

Labor Hours:	40	Actual Effective Working Hours:	33
		Potential Effective Working Hours:	33
Labor Cost:	1000	Effective Working Time per Labor Dollar:	0.03
Overhead:	800	Potential Effective Working Time	
Fee:	150	per Labor Dollar:	0.03

Total Cost:	1,950		

Lines of Code:	104	Cost per LOC:	19
Pages of Doc.:	3	Cost per POD:	650

Press Enter to continue...

Total Expenses for Activity

Activity: Software/Hardware System Analysis

Start Date: 05/14/91 End Date: 05/26/91

Labor Hours:	80	Actual Effective Working Hours:	66
		Potential Effective Working Hours:	66
Labor Cost:	2000	Effective Working Time per Labor Dollar:	0.03
Overhead:	1600	Potential Effective Working Time	
Fee:	300	per Labor Dollar:	0.03

Total Cost:	3,900		

Lines of Code:	208	Cost per LOC:	19
Pages of Doc.:	6	Cost per POD:	650

Press Enter to continue...

Expenses and Productivity for the Week Ending 06/24/91

Activity: Analyze Reqs of Functions and Data

Start Date: 06/10/91 End Date: 06/28/91

Labor Hours:	80	Actual Effective Working Hours:	63
		Potential Effective Working Hours:	63
Labor Cost:	2000	Effective Working Time per Labor Dollar:	0.03
Overhead:	1600	Potential Effective Working Time	
Fee:	300	per Labor Dollar:	0.03
Total Cost:	3,900		

Lines of Code:	197	Cost per LOC:	20
Pages of Doc.:	6	Cost per POD:	650

Press Enter to continue...

Total Expenses for Activity

Activity: Analyze Reqs of Functions and Data

Start Date: 06/10/91 End Date: 06/28/91

Labor Hours:	160	Actual Effective Working Hours:	126
		Potential Effective Working Hours:	126
Labor Cost:	4000	Effective Working Time per Labor Dollar:	0.03
Overhead:	3200	Potential Effective Working Time	
Fee:	600	per Labor Dollar:	0.03
Total Cost:	7,800		

Lines of Code:	394	Cost per LOC:	20
Pages of Doc.:	12	Cost per POD:	650

Press Enter to continue...

Expenses and Productivity for the Week Ending 06/24/91

Activity: Identify User Interface Reqs

Start Date: 06/10/91 End Date: 06/21/91

Labor Hours:	40	Actual Effective Working Hours:	33
		Potential Effective Working Hours:	33
Labor Cost:	1000	Effective Working Time per Labor Dollar:	0.03
Overhead:	800	Potential Effective Working Time	
Fee:	150	per Labor Dollar:	0.03

Total Cost:	1,950		

Lines of Code:	104	Cost per LOC:	19
Pages of Doc.:	3	Cost per POD:	650

Press Enter to continue...

Total Expenses for Activity

Activity: Identify User Interface Reqs

Start Date: 06/10/91 End Date: 06/21/91

Labor Hours:	80	Actual Effective Working Hours:	66
		Potential Effective Working Hours:	66
Labor Cost:	2000	Effective Working Time per Labor Dollar:	0.03
Overhead:	1600	Potential Effective Working Time	
Fee:	300	per Labor Dollar:	0.03

Total Cost:	3,900		

Lines of Code:	208	Cost per LOC:	19
Pages of Doc.:	6	Cost per POD:	650

Press Enter to continue...

Expenses and Productivity for the Week Ending 06/24/91

Activity: Define Functions and Data

Start Date: 06/10/91 End Date: 07/05/91

Labor Hours:	40	Actual Effective Working Hours:	33
		Potential Effective Working Hours:	33
Labor Cost:	1000	Effective Working Time per Labor Dollar:	0.03
Overhead:	800	Potential Effective Working Time	
Fee:	150	per Labor Dollar:	0.03

Total Cost:	1,950		

Lines of Code:	104	Cost per LOC:	19
Pages of Doc.:	3	Cost per POD:	650

Press Enter to continue...

Total Expenses for Activity

Activity: Define Functions and Data

Start Date: 06/10/91 End Date: 07/05/91

Labor Hours:	80	Actual Effective Working Hours:	66
		Potential Effective Working Hours:	66
Labor Cost:	2000	Effective Working Time per Labor Dollar:	0.03
Overhead:	1600	Potential Effective Working Time	
Fee:	300	per Labor Dollar:	0.03

Total Cost:	3,900		

Lines of Code:	208	Cost per LOC:	19
Pages of Doc.:	6	Cost per POD:	650

Press Enter to continue...

Total Project Expenses

Report Date: 06/24/91

Labor Hours:	884	Actual Effective Working Hours:	701
		Potential Effective Working Hours:	701
Labor Cost:	22,100		
Overhead:	17,680	Effective Working Time per Labor Dollar:	0.03
Fee:	3,315	Potential Effective Working Time per Labor Dollar:	0.03
Total Cost:	43,095		

Lines of Code:	2191	Cost per LOC:	20
Pages of Doc.:	64	Cost per POD:	673

Percent of Budget Spent: 6%
Percent of Schedule Spent: 10% Percent of Product Completed: 13%

Press Enter to continue...

Enter the number of weeks you wish to advance Today's Date or zero to exit without advancing the date:

4

Note: Processing will continue until all weeks have passed.

Total Project Expenses

Report Date: 07/22/91

Labor Hours:	1404	Actual Effective Working Hours:	1137
		Potential Effective Working Hours:	1137
Labor Cost:	35,100		
Overhead:	28,080	Effective Working Time per Labor Dollar:	0.03
Fee:	5,265	Potential Effective Working Time per Labor Dollar:	0.03
Total Cost:	68,445		

Lines of Code:	3043	Cost per LOC:	22
Pages of Doc.:	89	Cost per POD:	769

Percent of Budget Spent: 10%
Percent of Schedule Spent: 16% Percent of Product Completed: 18%

Press Enter to continue...

Enter the number of weeks you wish to advance Today's Date or
zero to exit without advancing the date:

14

Note: Processing will continue until all weeks have passed.

Software Design Review Weekly Report

Pages of Design Documentation Reviewed: 98.40

Errors Removed From Design: 61.50

Note: Software Design Review consists of design inspections performed according to the procedures outlined by M.E. Fagan in the paper 'Design and code inspections to reduce errors in program development' published in the IBM Systems Journal, Vol. 15., 1976.

Press Enter to continue...

Software Design Review Weekly Report

Pages of Design Documentation Reviewed: 86.19

Errors Removed From Design: 53.86

Note: Software Design Review consists of design inspections performed according to the procedures outlined by M.E. Fagan in the paper 'Design and code inspections to reduce errors in program development' published in the IBM Systems Journal, Vol. 15., 1976.

Press Enter to continue...

Expenses and Productivity for the Week Ending 10/21/91

Activity: Internal Review of Software Design

Start Date: 10/07/91 End Date: 10/18/91

Labor Hours:	120	Actual Effective Working Hours:	120
		Potential Effective Working Hours:	120
Labor Cost:	3000	Effective Working Time per Labor Dollar:	0.02
Overhead:	2400	Potential Effective Working Time	
Fee:	450	per Labor Dollar:	0.02
Total Cost:	5,850		
Lines of Code:	0	Cost per LOC:	0
Pages of Doc.:	0	Cost per POD:	0

Press Enter to continue...

Total Expenses for Activity

Activity: Internal Review of Software Design

Start Date: 10/07/91 End Date: 10/18/91

Labor Hours:	240	Actual Effective Working Hours:	240
		Potential Effective Working Hours:	240
Labor Cost:	6000	Effective Working Time per Labor Dollar:	0.04
Overhead:	4800	Potential Effective Working Time	
Fee:	900	per Labor Dollar:	0.04
Total Cost:	11,700		
Lines of Code:	0	Cost per LOC:	0
Pages of Doc.:	0	Cost per POD:	0

Press Enter to continue...

Expenses and Productivity for the Week Ending 10/28/91

Activity: Module Development

Start Date: 10/21/91 End Date: 02/21/92

Labor Hours:	160	Actual Effective Working Hours:	111
		Potential Effective Working Hours:	111
Labor Cost:	4000	Effective Working Time per Labor Dollar:	0.03
Overhead:	3200	Potential Effective Working Time	
Fee:	600	per Labor Dollar:	0.03
Total Cost:	7,800		

Lines of Code:	333	Cost per LOC:	23
Pages of Doc.:	0	Cost per POD:	0

Press Enter to continue...

Total Project Expenses

Report Date: 10/28/91

Labor Hours:	4444	Actual Effective Working Hours:	3547
		Potential Effective Working Hours:	3547
Labor Cost:	111,100		
Overhead:	88,880	Effective Working Time per Labor Dollar:	0.03
Fee:	16,665	Potential Effective Working Time	
		per Labor Dollar:	0.03
Total Cost:	216,645		

Lines of Code:	9532	Cost per LOC:	23
Pages of Doc.:	267	Cost per POD:	811

Percent of Budget Spent: 34%
Percent of Schedule Spent: 40% Percent of Product Completed: 57%

Press Enter to continue...

Enter the number of weeks you wish to advance Today's Date or
zero to exit without advancing the date:

20

Note: Processing will continue until all weeks have passed.

Total Project Expenses

Report Date: 03/16/92

Labor Hours:	8124	Actual Effective Working Hours:	6367
		Potential Effective Working Hours:	6367
Labor Cost:	203,100		
Overhead:	162,480	Effective Working Time per Labor Dollar:	0.03
Fee:	30,465	Potential Effective Working Time	
	---------	per Labor Dollar:	0.03
Total Cost:	396,045		

Lines of Code:	16552	Cost per LOC:	24
Pages of Doc.:	267	Cost per POD:	1483

Percent of Budget Spent: 63%
Percent of Schedule Spent: 73% Percent of Product Completed: 100%

Press Enter to continue...

Enter the number of weeks you wish to advance Today's Date or zero to exit without advancing the date:

13

Note: Processing will continue until all weeks have passed.

Software Review Weekly Report

Lines of Code Reviewed: 3280.

Errors Removed From Code: 34.44

Note: Software Review consists of design inspections performed according to the procedures outlined by M.E. Fagan in the paper 'Design and code inspections to reduce errors in program development' published in the IBM Systems Journal, Vol. 15., 1976.

Press Enter to continue...

Software Review Weekly Report

Lines of Code Reviewed: 3280.

Errors Removed From Code: 34.44

Note: Software Review consists of design inspections performed according to the procedures outlined by M.E. Fagan in the paper 'Design and code inspections to reduce errors in program development' published in the IBM Systems Journal, Vol. 15., 1976.

Press Enter to continue...

Software Review Weekly Report

Lines of Code Reviewed: 772.

Errors Removed From Code: 8.10

Note: Software Review consists of design inspections performed according to the procedures outlined by M.E. Fagan in the paper 'Design and code inspections to reduce errors in program development' published in the IBM Systems Journal, Vol. 15., 1976.

Press Enter to continue...

Total Errors Found and Corrected by Phase

	For the Past Week	Total Project
System Definition	8	104
Software Design	20	260
Software Development	12	156
Total	40	520
Lines of Code Tested	1,220	15,860
Cost of Error Removal	22,892	297,596

Press Enter to continue...

Client Acceptance Test Results

The product delivered to the client consists of 16552 lines of code.

Product functionality was found to be complete.

The results of acceptance testing shows a possible 1.87 errors per KLOC remaining in the product which is acceptable.

Based on the information above, the client has accepted the product. Congratulations.

Press Enter to Continue...

Total Project Expenses

Report Date: 06/22/92

Labor Hours: 22152 Actual Effective Working Hours: 19718

Potential Effective Working Hours: 19718

Labor Cost:	553,696		
Overhead:	442,957	Effective Working Time per Labor Dollar:	0.04
Fee:	83,054	Potential Effective Working Time	
	---------	per Labor Dollar:	0.04
Total Cost:	1,079,707		

Lines of Code:	16552	Cost per LOC:	65
Pages of Doc.:	267	Cost per POD:	4037

Percent of Budget Spent: 172%

Percent of Schedule Spent: 96% Percent of Product Completed: 100%

Press Enter to continue...

VITA

Ms. Galbraith has ten years of experience in programming and software project management and is currently owner of Effective Software Solutions, Inc., a software design company servicing the Southeast. Prior to forming this company, she provided software engineering guidance and systems analysis support in her role as Program Manager with Integrated Computer Systems, Inc. She began her career as a programmer analyst and project leader with Science Applications International Corporation, where she worked with government and commercial clients to develop business software applications. Ms. Galbraith worked as a technical research assistant with Union Carbide Corporation while studying for a Bachelor's degree in Computer Science from The University of Tennessee, which she received in 1982.