To the Graduate Council:

I am submitting herewith a thesis written by Barbara A. Campbell entitled "Software Cost Estimating System." 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.

Terry Feagin, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

The Graduate School

SOFTWARE COST ESTIMATING SYSTEM

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Barbara A. Campbell

December 1983

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Terry Feagin who guided my research. My appreciation also goes to Dr. David Straight and Dr. Charles Pfleeger for serving on my committee. This work was supported by the Engineering and Computer Sciences Divisions of Union Carbide Corporation, Nuclear Division.

ABSTRACT

A study was made of computer software and factors which may influence the development and maintenance costs in terms of man hours. Data was gathered using surveys. The responses to the survey questions were quantized, a database was designed, and the data was entered into the database. Report generators were developed to output various data in a readable form.

An attempt was made to develop a reliable method of estimating man hours spent developing and subsequently modifying and enhancing software systems. A secondary goal was to show that the methods used in maintaining a system are as important as the initial development methods and that the development methods influence the maintenance effort. A group of attributes were selected which best applied to the installation where data was gathered. Among these, many have been studied previously and some have not, namely security classification and a breakdown of number of people per development phase, average years of experience of people during the different development phases, and the structured methodology of each phase.

Maintenance attributes included some development attributes as well as adding new ones. The maintenance of the structured development documentation was also considered.

TABLE OF CONTENTS

CHAPTER		PAGE
I.	HISTORICAL BACKGROUND	1
II.	FOUNDATIONS AND ASSUMPTIONS	18
III.	DEVELOPMENT OF A DATABASE	23
IV.	SOFTWARE COST ESTIMATING SYSTEM	25
V.	EXPERIMENTAL RESULTS	29
VI.	CONCLUSIONS	51
LIST OF REFERENCES		55
APPENDICES		60
APPENDIX A.	SURVEY FORM	61
APPENDIX B.	OBJECTIVES REPORT	68
APPENDIX C.	SOFTWARE	95
APPENDIX D.	GLOSSARY	113
VITA		115

LIST OF TABLES

TABLE		PAGE
1.	Numbered Attributes	30
2.	Correspondence Ratios for Development Attributes	31
3.	Development Conversion Coefficients	33
4.	Development Conversion Equation Errors	35
5.	Development Estimates	38
6.	Modification Conversion Coefficients	41
7.	Modification Conversion Equation Errors	43
8.	Modification Estimates	47

LIST OF FIGURES

FIGURE		PAGE
1.	Software Development Effort	2
2.	Software Life Cycle Effort	3
3.	Manpower Curve	6
4.	Graphical Representation of Gradient Curves	10
5.	Relative Cost per Routine	17
6.	Industry Averages	19

CHAPTER I

HISTORICAL BACKGROUND

Traditionally, software costs have been high, and they are becoming a higher percentage of total computer costs due both to rapidly decreasing hardware costs and increasing software costs. When considering the cost of software, maintenance costs must be considered as well as development or purchase costs.

Figures 1 and 2 show the relative percentages of effort and consequently the percentages of cost of the various stages of the software development cycle and the entire software life cycle, respectively. The majority of the cost involved in software comes from maintenance, 70 percent versus 30 percent for development. Software maintenance includes any work performed on a system after acceptance. It also includes any enhancements or modifications done to the system which are not part of the original objectives and requirements.

Software groups have been plagued with the problems of consistently running over budget and past deadlines. It has become imperative for a reliable method of software cost estimating to be developed. Current methods are based on intuition and incomplete software cost databases.

There are several traditional methods of software cost estimating: top-down estimating, similarities and differences estimating, ratio estimating, standards estimating, quantitative estimating, constraint estimating, and bottom-up estimating. It has been suggested by Wolverton

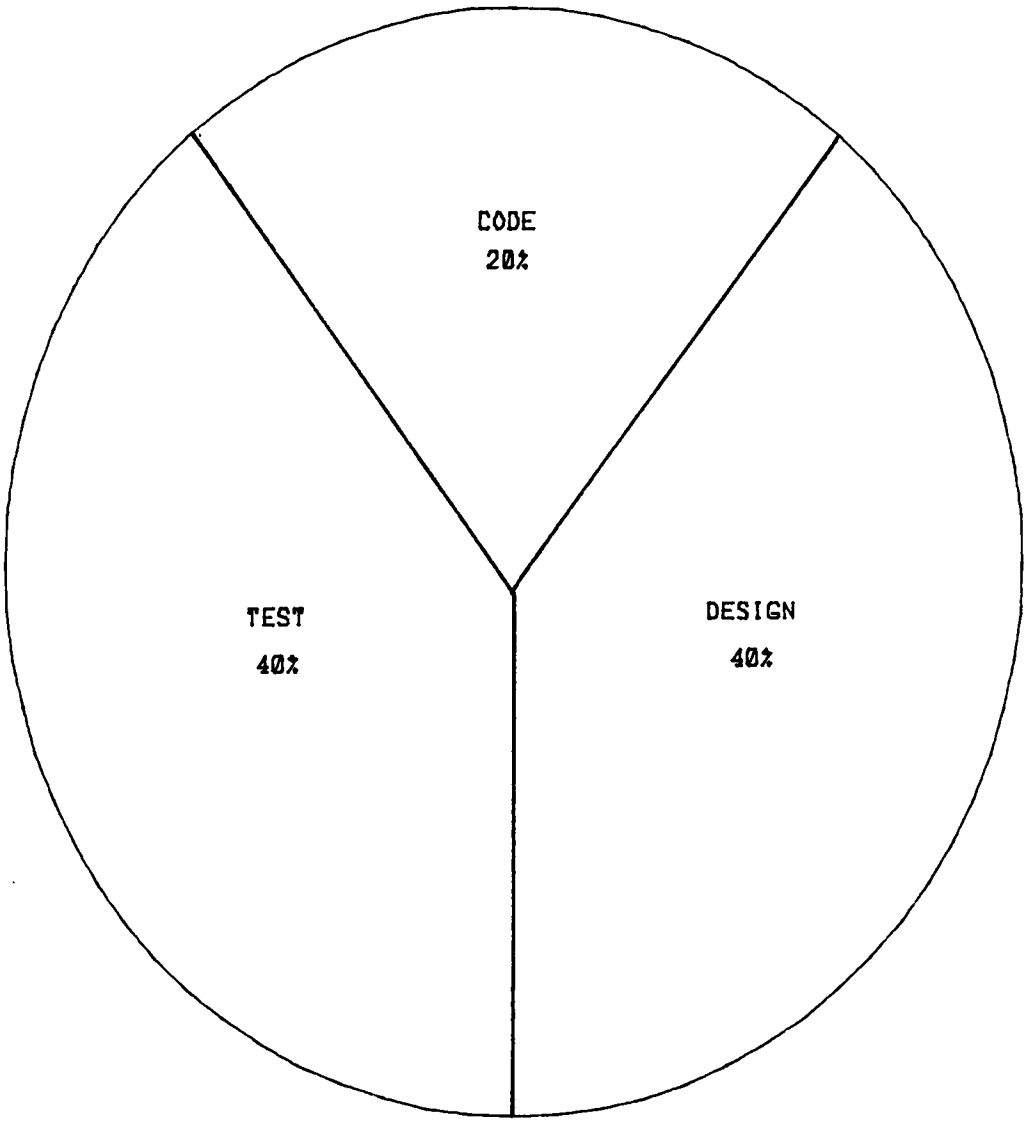


Figure 1. Software Development Effort

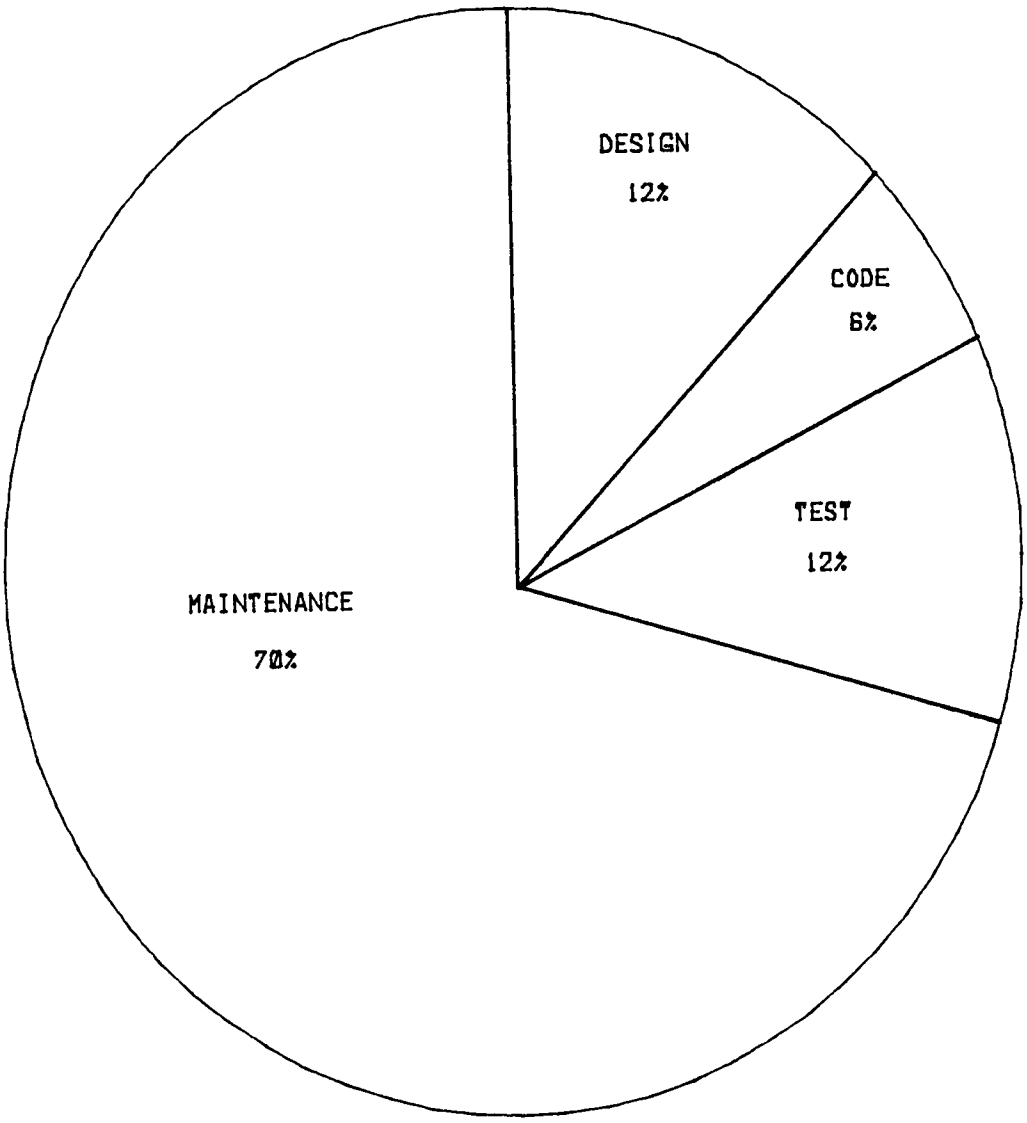


Figure 2. Software Life Cycle Effort

[39] that the general principles involved in pricing research and development efforts of any kind apply to software development as well.

In top-down estimating, estimates are based on the cost of a previous, similiar project coupled with "informed opinion" of the costs of the differences between the systems. The problem arises in overlooking some special differences which may be particulary complex. Similarities and differences estimating is much the same, but the systems are broken down into smaller, simpler, work units which are then compared and estimated separately. Ratio estimating has the analyst estimating sizes of individual modules, and their relative complexity. A developed, historical cost database is then used to produce an estimate. A valid data base must exist for accurate estimating to occur.

In standards estimating, the estimator relies upon standards of performance which have been systematically developed. The development of custom software is most likely not performed repeatedly, so no standards have been developed. Quantitative estimating uses estimates of specified variables to adjust a general estimate. Contraint estimating assumes the schedule to be fixed and adjusts the specifications to fit the other constraints of schedule, man-power, money, etc. Bottom-up estimating, which is used most often, breaks the job into small work packages which are basic enough to give a reasonably clear idea of what resources will be needed to complete each. The problem arises in combining all the work and all the cost estimates, to look at the job as a whole.

Two new methods are gaining in popularity, smoothing and extrapolation, and the use of algorithms. Smoothing and extrapolation

incorporate a past history of trends in the percentages of time taken for each step of a new project, so that after the initial analysis is done, the rest of the work effort can be predicted. There are several algorithms in use at this time, such as the Norden/Rayleigh manpower equation, and a weighted variable productivity index summation.

Software development and code production rates are dynamic. Some software state variables are

1. C_k the state-of-technology
2. K the applied effort
3. t_d the development time
4. t the independent variable time.

Putnam [30] has shown that software systems follow a lifecycle pattern which is well described by the Norden/Rayleigh manpower equation

$$y = 2 K a t e^{-at^2}, \qquad (1.1)$$

(where $a = 1/(2 t_d^2)$, t_d is the time at which y is maximum, and K/t_d^2, a function of these, is the difficulty). K is the time under the curve from t = 0 to t = infinity, and is the life cycle effort in man years.

Norden found that projects are composed of cycles of planning, design, modeling, release, and product support. By graphing the cycles and their respective time relationships, and adding them together, a profile of an entire project may be produced (Figure 3). When the model was tested using 50 large-scale business type applications systems, the aggregate followed this life cycle model very well [30].

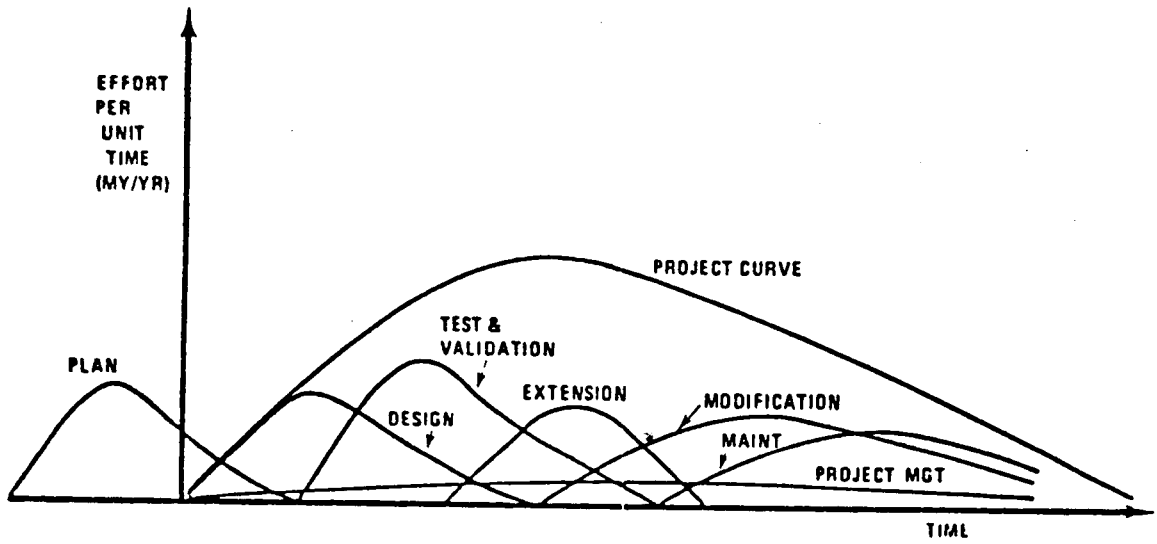


Figure 3. Manpower Curve

Note that this manpower curve is controlled by management. Noise may occur due to inadequate specifications, changes in requirements, human communications, lack of management. The noise component may be as high as plus or minus 25% of the expected value during the development effort (rising part of the curve) due to these factors.

The definite integral of (1.1) is

$$y = K\ (1 - e^{-at^2})\ \text{man years} \tag{1.2}$$

which is the cummulative number of man years spent on the system at any time t. K and a can take on a range of values. K is the total life cycle man years of a process and a is the shape paramenter. By making a substitution for a, the Rayleigh equation can be re-written

$$y = K/t_d^2\ t\ e^{-t^2/2t_d^2} \tag{1.3}$$

where,

$K/t_d^2\ t$ has units of power (effort per time), and

K/t_d^2 has units of force (manpower per time).

By taking logarithms and thereby linearizing,

$$\ln\ (y/t) = \ln\ (K/t_d^2) + (-1/2t_d^2)\ t^2 \tag{1.4}$$

When ln (y/t), the natural logarithm of manpower divided by the corresponding elapsed time, is plotted against t^2, elapsed time squared, a straight line is produced in which ln (K/t_d^2) is the intercept, and $-1/2t_d^2$ is the slope. For some 100 systems [30], it was found that K/t_d^2 represented the difficulty of a system in terms of the programming effort and the time to produce it.

The manpower rate involves the inter-communication law [4]:

$$\text{complexity} = N\ [\ (N - 1)\ /\ 2] \tag{1.5}$$

where N is the number of people that have to communicate. Note that

when development time is shortened, difficulty increases dramatically, usually to an impossible level. When people are added to a project, complexity increases.

For each software house, a set of lines can be found where each line represents a type of system or a class of work, and portrays a trace of a constant magnitude of the difficulty gradient. The difficulty gradient is the rate of change of the difficulty over time. Finding these lines for a given installation is a problem.

This implies that, for a given software organization, as the system size increases, the development time must increase so as to remain on the trace of a line having constant magnitude of the gradient defined by

$$K/t_d^3 \tag{1.6}$$

Adding people to accelerate a project can be useful, but only until the gradient condition is reached and at a very high cost.

Putnam finds that the size of a product in source statements is a function of the effort and the development time,

$$S_s = C_k K^{1/3} t_d^{4/3} \tag{1.7}$$

The constant, C_k, is determined by technical parameters such as machine throughput, structured programming, and on-line interactive job submission. Equation (1.7) then is a management equation for software system building, in which, as the use of new technology increases, C_k increases and throughput, S_s, increases. However, the number of source statements in the final product must be known for this to have any bearing on an initial estimate.

Putnam defines productivity as

$$C_k D^{-2/3} \tag{1.8}$$

where D equals the difficulty of the system, K/t_d2. Constant productivity implies constant difficulty. Since difficulty is a measure of the software work to be done this seems unreasonable. The difficulty should logically decrease as the amount of work left to be done decreases. If we change the system characteristics during development then we change the difficulty, and therefore the productivity will also change. Productivity is a rate, and it is therefore fallacious to think in terms of just adding time to an estimate in response to a change in requirements or specifications, when actually, it is the change in difficulty which effects a change in the productivity. The problem now is to determine K and t_d as early in the process as possible--during the analysis phase, as requirements and specifications are being introduced.

One method is to use a graphical representation of equation (1.7) as in Figure 4 [31]. Assume that C_k is known for a specific software installation; then choose a range for the size of the system and a range for the desired development time. Although this method still uses a significant amount of guess work, it does involve some theoretical cost estimating practices.

Another method is to combine prior history with regression analysis to relate management parameters to that set of system characteristics which can be determined during anaysis. Here, an attempt is made to use prior history with correspondence ratios and weighted, valued system attributes fit to a curve to determine difficulty which directly relates to productivity.

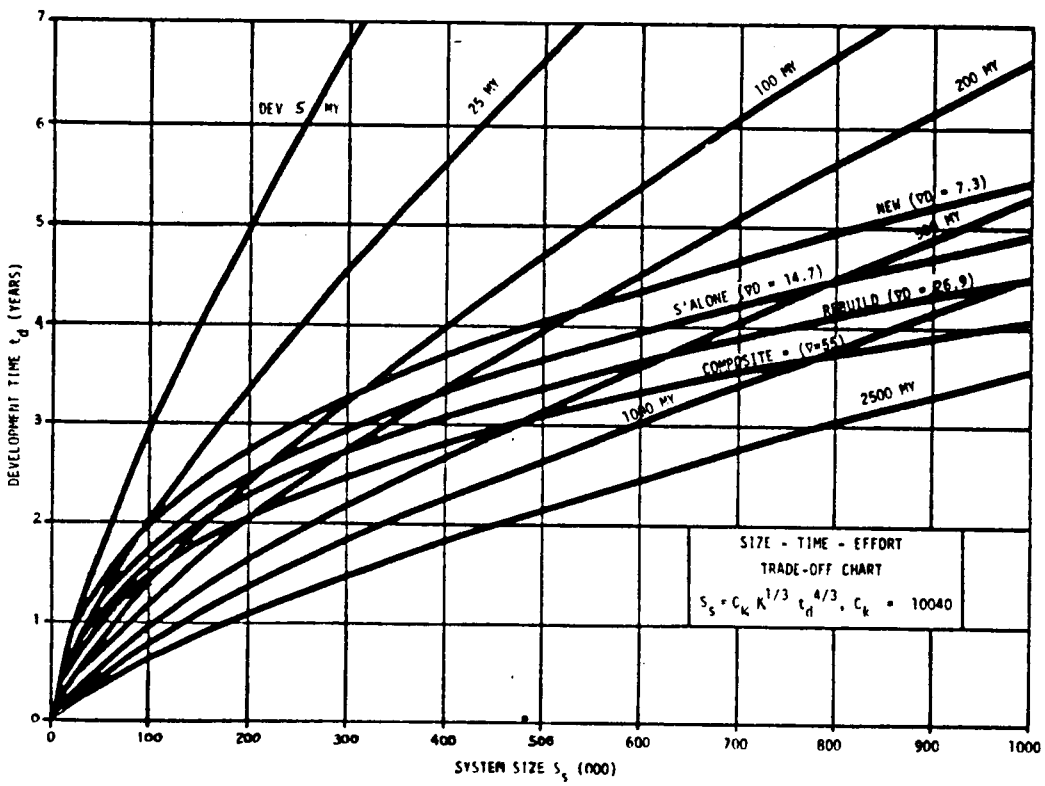


Figure 4. Graphical Representation of Gradient Curves

K and t_d are not independently linearly related to system attributes such as number of files, number of reports, and number of application programs. However, $K/t_d2 = D$ is linear with the number of files, number of reports, and number of application subprograms, both individually and jointly. The number of files and the number of reports are highly redundant since data would not be stored in a file unless it were going to be reported. Therefore, a good estimator can be found using just the number of application subprograms and either the number of reports or the number of files. These variables can be estimated during the requirements specification process to within 10 to 15 percent. For business applications, the number of files and applications subprograms work very well as estimators. For other applications, other characteristics should be considered along with these.

A numerical procedure to obtain estimators for the various functions of K and t_d,

K/t_d3 = difficulty gradient

K/t_d2 = difficulty

K/t_d = manpower average

K = total effort

is to use multiple regression analysis and data determined from past history of the software house.

Using an equation of the form

$$a_2 x_2 + a_3 x_3 = K/t_d^i \qquad (1.9)$$

where a_2 and a_3 are constant coefficients of x_2, representing either the number of files or reports, and x_3, the number of applications programs, respectively, a set of simultaneous equations can be formed, using real

data. This can be done for each K/t_d^i term. These may be solved using matrix algebra. The a_i estimators are specific to each software house and depend upon the application. Once they are determined for all terms, and estimates for the number of reports or files, and application subprograms are made, estimates for K and t_d may be made by solving any subset of the simultaneous equations. This method yields more reliable estimates for large systems because data on larger systems is more consistent. Extrapolation outside the range of data used to generate the estimators will yield poor results. Therefore, it would be advantageous to set up systems of equations for each class and several size groups of systems, but this defeats the idea of estimating a system before the size is known.

We have determined how to estimate software costs given a productivity rate and development time constraints and difficulty of a system. Development time constraints are controlled by management. We need reliable algroithms to determine productivity rates and difficulty. Let's start with productivity.

Measuring productivity is a difficult task as we do not want to measure the productivity of an individual but rather of a software house.

Walston and Felix [37], describe a database developed at IBM to store and report data relating to software measurement and estimation. Their studies analyze variables independently, not taking into account correlations or interrelated effects. The results are still significant and contribute greatly to the knowledge base in the software cost estimating field.

Twenty-nine variables were used to compute productivity indexes for fifty-one projects using

$$I = \text{sum } (W_i X_i) \tag{1.10}$$

where,

I = productivity index for a project,

W_i = variable weight,

X_i = question response (1, 0, -1) depending upon the increasing, nominal or decreasing effect of the variable or productivity.

The key here then is in finding the W_i's and determining how to use I in a cost estimate.

The W_i's are calculated

$$W = (\log (\text{delta } p/\text{nom}(p)))/2 \tag{1.11}$$

where,

delta p = difference in productivity between the highest mean and lowest mean of an attribute

nom (p) = the mean for the nominal case.

It is interesting to note that the study also suggests that "productivity is highest when there is no original or reused code . . . . As the percentage of reused code grows, the expected productivity decreases." This agrees with Putnam's findings, where he discusses entropy. In fact, by using two different approaches, the same conclusions were deduced, namely, development application code which is dependent upon existing application code has the highest entropy, the greatest difficulty gradient and, hence, the lowest productivity rate.

It has been found that the number of pages of documentation varies

directly as the number of lines of source code. However, this does not give any indication as to how documentation affects the life cycle costs of a system. And the effect of documentation on costs has not, as yet, been studied.

Wolverton [39], explains how TRW developed a cost estimation algorithm based on the assumption that costs vary proportionally with the number of instructions. The method uses the variables: the estimated number of object instructions per identified module, a classification of the function of the module, and the relative difficulty to give an estimate from a historic database in dollars per instruction. Software categories used were: non-time-critical control routines controlling execution flow; input and output routines transferring data between the computer and the world outside the computer; pre- and post-algorithm processing routines which manipulate data for subsequent processing or output; algorithm routines performing mathematical calculations; data management routines transferring data within the computer system; and time-critical control routines using highly optimized machine dependent code. This was implemented using the dual type attributes which relate to these catagories.

The difficulty and number of instructions must be estimated. This may be done based on the machine used, the language used, similarity with previously completed modules, and the person doing the job. There is conflict among experts as to whether or not valid performance standards can be set for software development. It is important to allow a sufficient degree of freedom to accommodate a learning experience and individual differences in performance. Standards of performance can be

used in software development as adequately as they can be used in any other developmental work. For this reason a break down in experience levels was used in this study.

From these, TRW comes up with a manpower estimate which is then broken down into a distribution through the stages of the project using the standard 40-20-40 percent levels of analysis/design-code-test respectively.

An advantage of this system is that by making an estimate per routine, the cost estimate may reflect the cost of a specific requirement to the extent that routines may be traced back to requirements. This may aid the customer in deciding upon the ultimate set of requirements, and it should also aid in prioritizing requirements. It does not however give any consideration to the integration costs of the system.

It is necessary for each installation to choose a set of measurable variables that influence the costs of their software products. Wolverton identifies the following:

1. learning and orientation
2. analysis and design
3. coding
4. test and checkout
5. verify/quality
6. system integration and test
7. documentation

Some analysis that has already been done interprets observations concerning size of routines, programmer experience, and difficulty.

Figure 5 shows data from an analysis by Lulejian and Associates [39], under contract to the U.S. Air Force, analyzing 88 routines developed by TRW. The only conclusions are that larger routines cost more and that a linear fit to data is not very good. It might be inferred from this data that the factors influencing the cost of software development are much more numerous and inter-related than this analysis explored.

Producing real-time (or time-critical) software is more costly than nonreal-time software. From analysis of one software system, programmer experience, and supervisor's rating of the programmer seemed to make no difference in the cost of the system [39]. A measure of quality of the software was not made, so the question arises as to whether the end product was of higher quality.

With respect to completeness of problem formulation, nothing takes the place of competence and communication in the understanding of the customer's requirements. It has been shown that thorough and continuous involvement of the customer and superior documentation with regular review by knowledgeable technical staff has a positive influence on the quality and productivity associated with a project [39]. Quality of software and problem formulation are difficult to measure. It may be possible that the ease or low cost of maintenance is an indication of high quality software. Certainly it would seem that thorough formulation should have a decreasing effect on maintenance costs, as the customers' desires can be included in the original development effort, and enhancements would less likely be required.

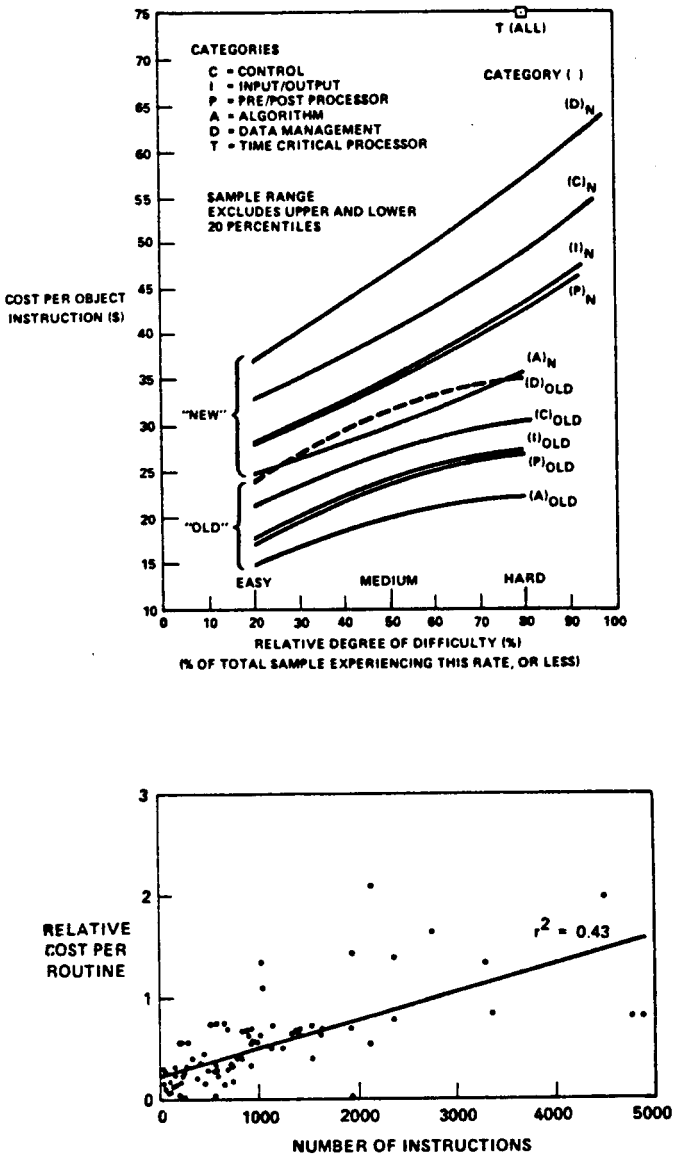


Figure 5. Relative Cost Per Routine

CHAPTER II

FOUNDATIONS AND ASSUMPTIONS

Studies by industry and DoD have concluded that there are no simple, universal rules for predicting software costs accurately. This is partly due to the poor historical databases of software costs. Turnover time for an average programmer is shown to be about 3 years, and this has a significant effect on productivity data.

There is no universal model. Industry-wide productivity rates in terms of program size and complexity have been studied. Figure 6 shows the averages. Some of the questions being asked by cost specialists are: Have we identified all the parameters needed in our cost database? Are the causal relationships understood, and can they be isolated from their effects? How does complexity cause increases in cost and how can management effectively detect and deal with this?

Several assumptions are made in this thesis based upon other researchers' findings in the software cost estimating field and as a result of the particular circumstances of the facility where the data was gathered. Also, this thesis reports on the collection and testing of certain data which would be irrelevant to many other software groups. As can be seen by the results of the surveys, the sampling covered a very broad range of applications, techniques, experience, and system size—maybe too broad a range.

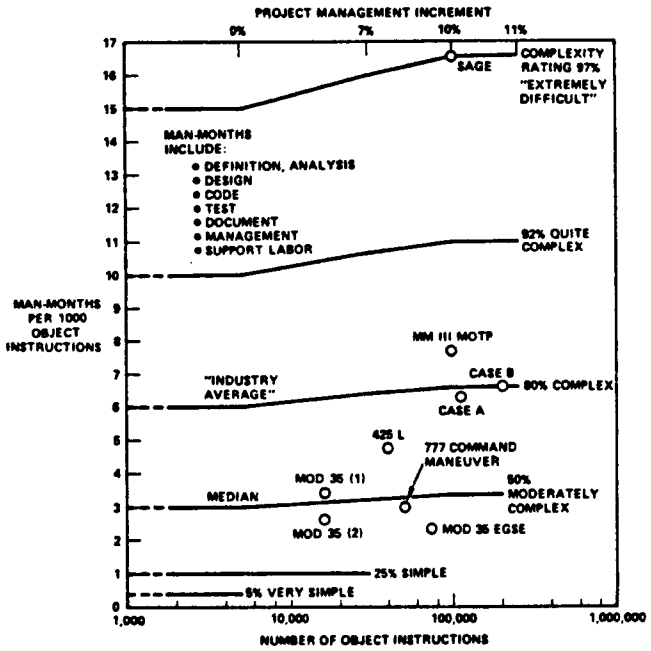


Figure 6. Industry Averages

The type of work done at the installation where the data was gathered is limited to in-house software. The programs do not need to be as general as those of a software group which develops and maintains only one system to be used by many customers. Data gathered by survey for this study covered systems developed for a single customer and maintained for that one customer.

Studies by other researchers have been unable to come up with conclusive evidence relating software costs with many quantitative characteristics of the software. This is due to several factors. It is difficult to quantize software characteristics such as structured analysis, design, code, management techniques, software quality and size, the language used, etc. Not only are these attributes difficult to measure, but it is very difficult to gather an adequate amount of accurate data. The use of information from time keeping is inadequate due to the methods of time keeping and the lack of tracking all expenses, including overhead and special accounts. Another problem is the subjectiveness of the data gathering method. Someone must decide what is structured and what is not and where something falls in between. If a single interviewer is used, the time involved for that one person would be exorbitant and inevitably, the interviewer's opinions would change with the exposure to many different systems. There also could be a risk of bias by the interviewer--since software applications are so broad no one person has the expertise to compare and judge the various languages, methods, and applications, fairly. Data is spread so sparsely, with so many inter-related variables that, when trends are seen, it is difficult

to say whether they are due to inadequacies of data gathering and availability or whether they are true trends.

So, research thus far has had to struggle with the problems of how to gather data, getting enough data, quantizing it, measuring it and then using it. There will be several ideas incorporated into this analysis to aid in the development of a quantitative system. These ideas are based on other research and trends.

Looking first at programmer performance, the difference in performance, in this case, production, between a high level and a low level may be as high as 28 times greater for the high level performer [11]. The difference may be attributable to training, selection, and placement. This implies a heavy dependency upon management.

Productivity is directly related to the complexity of the task. And, although this study does not touch on the psychological ramifications of complexity, it is well to note that the psychology of software engineering is a study in itself. Halstead [11] measures the complexity, E, of a software module

$$E = \frac{n_1 N_2 (N_1+N_2) \log_2 (n_1+n_2)}{2 n_2} \qquad (2.1)$$

where,

n_1 = number of unique operators

n_2 = number of unique operands

N_1 = total frequency of operators

N_2 = total frequency of operands.

This, however, is only the complexity of a single module or more accurately, submodule. A greater complexity comes in integrating all the submodules into modules and then all the modules into a system. So

it would follow that the more cohesive the submodules are, the less complex they will be individually, which will in turn have a decreasing effect on the complexity of the whole system. The complexity of a modification increases with the number of new lines inserted into the code. And of course, the more complex the original code, the more complex will be any modification to it. Sheer length has also been found to be moderately related to complexity. The size of a module can implicitly give some indication as to the complexity of the module. A smaller module has less chance of having a large E value for the simple reason that less is being done.

Maintenance of any naturally structured module with documentation is simpler than maintenance of an unstructured module [11]. A set of types of modifications were developed to allow for the variences in difficulty of making a modification.

CHAPTER III

DEVELOPMENT OF A DATABASE

A structured approach was used to develop this software system. The difficulty came in that I was my own customer. So, I spent several months researching existing studies as well as interviewing software professionals, to find out what some of the constraints and requirements for developing an estimating system might be. A survey form was developed and walked-through by a group of computer professionals, modifications were made and a final survey form (see Appendix A) with instructions was produced.

Since terminology and standards in software engineering are somewhat ambiguous, the surveys were presented to groups of 5 to 7 programmers at a time. Some simple standards were explained and terms defined. I was available at all times to answer questions. Each group of programmers typically represented one departmental section or group with one job function. In this way, when there were questions about group projects, in most cases the responsible programmer could consult with peers and project team members. A maximum of 30 minutes was given to fill out the forms.

It is recognized that some parts of the form are biased because Yourdon terminology was used and not all programmers are familiar with this development methodology. However, some basis had to be chosen, and as Yourdon's method seems common it was adopted. Once the data was

gathered, a system had to be developed to store the data on the computer. Following a structured approach to software development, an objectives report was written, the contents of which are contained in Appendix B.

Following completion of this report, the decision came as how best to implement the system: what computer to use, what software language, what software tools or packages to use?

A PDP-11/34 was chosen due to availability and familiarity by this programmer. It was decided to use the DEC Datatrieve database system to store and report data due to its ease of use and numerical abilities. Record, database layout and example report generators are included in Appendix C. Subroutines for record extraction for use in Fortran programs are available through DECUS--this also was an influencing factor since it might facilitate the numerical analysis of data.

The system is a dynamic system where, as old data is modified and new data is introduced for analysis, it can be used to develop a new estimating algorithm very easily.

CHAPTER IV

SOFTWARE COST ESTIMATING SYSTEM

A method, most comparable to the one developed at IBM [34], was developed, with the following modifications: Twenty nine attributes were used for development projects and, on questions which could be quantized, a scale of six gradations was used, allowing for a more complex relationship curve. Also, and of significance, a study of modifications was added, which included eleven attributes, as well as the information retained from the development phase.

The first question to be answered was whether to treat all systems equally or to divide them into subgroups, all members of a subgroup having some common attribute. And then, if the subgroup method were adopted, what would be the determining attribute. Another issue addressed was that of inter-relatedness of attributes. Based on previous research, evidence points towards certain types of software systems being inherently more "difficult" than others. The question arises as to what makes them more difficult and if this particular attribute were occasionally present in a different type of system would it have a quantitatively similiar influence on the difficulty of the system.

The type of system was chosen as a dominant attribute and the systems were grouped accordingly. Each occurance of a high reading, four through six, for an attribute was tallied within each type group, and the percentage of systems of a particular type with a high value for an

Software Cost estimating System
Thesis 83.C.254

ılated. If 75 percent or higher of the systems cross
› attributes, it was said there was a positive corre-
ɔent or less of the systems cross matched then there
ɛelation or inverse correlation. In either case the
ɔreased by following the assumption that the correlat-
actored in at full weight. Hence the current attri-
have been factored in to some degree. That degree
distance from zero correlation or 50 percent for that
istance is then subtracted from 1 (100 percent) to de-
tage of weight to use. Note that for zero correla-
of the weight would be used.

ɪtely large data base all ratios would be used. In
:o the limited number of entries in the data base, values of one and zero are ignored as they tend to imply that little or no data was available for that particular type of system. A zero would imply that no data was available and a one would imply that only one set of data was taken for that type. As the data base grows this special treatment may be eliminated. It should be noted that in some cases there may be correlation among three or more attributes. This information is then used in finding the relative influence of the attributes on a project, by distributing the effect among all.

An algorithm similiar to that developed at IBM was used to develop initial, weighting factors. On quantitative questions the survey form had six ratings from high = 1 (agreement with the statement) to low = 6 (disagreement with the statement), plus a catagory for not applicable, of which, the value varies with the question. In some cases, not

applicable represents a zero rating and in other cases it will represent the extreme, completely disagree.

The following describes the weighting equation used:

$$W_i = \frac{\text{sum } (k=1,n) \text{ of attribute}_{i_k} / n}{\text{sum}(j=1,N) \text{ of attribute}_{i_j} / N} \qquad (4.1)$$

where,

W_i = weighting factor of attribute i

n = number of projects with attribute i, n =< N

attribute_{i_k} = number of man-hours required for project k with attribute i, k =< N, divided by the number of lines of code (normalized man-hours)

N = total number of projects

attribute_{i_j} = number of normalized man-hours required for project j

Note that the denominator (sum (j=1,N) of attribute $_{i_j}$/N) is merely the average number of normalized man-hours for all projects surveyed. This number will then be used for all attributes. Inter-correlating factors are used later to avoid redundant weighting. Due to the large range of projects used, with man-hours ranging from 450 to 35,000 and size ranging from 100 to 350,000 lines of code, merely using man-hours was an imperfect way of determining weight. Modification to "normalized" man-hours was adopted to help minimize the error introduced by the large range. Consequently, the number of man-hours per line of code was used.

By using this system, positive and negative effects are built in by producing fractional weights which represent factors having an increasing or reducing effect on development time and having numbers greater than one which will increase an estimate accordingly.

To calculate a new set of correspondence ratios, program CORESP may be run. To calculate a new set of development coefficients, program WEIGHT may be run. To calculate a development estimate and obtain a report of all the above with error reporting included, program DEVEST may be run. Program MWGHT calculates maintenance attribute coefficients and program MODIF calculates modification estimates. No correspondence ratios were calculated for modifications.

Originally, only the weights would be used as multipliers for an attribute. But as data became available and was studied it appeared that a method was needed to smooth the results and help decrease some of the error attributable to noise in the data. A weighted polynomial curve fit was used in an attempt to accomplish this.

CHAPTER V

EXPERIMENTAL RESULTS

Table 1 contains a numbered list of the attributes. The first report produced, Table 2, was an analysis of corresponding attributes. The primary attribute used was type. Systems had two type fields, a primary and a secondary, though they were treated equally, for correspondence purposes. The table shows the attribute numbers in the left column. The other columns are headed by the system type and contain the decimal fraction of systems of that type having the respective attribute with a value of four or greater. (Attributes one and two are the system types and were not reported for correspondence purposes although it would be interesting to know which types of systems over-lap the most).

No data was taken involving operating systems, so this type of system was eliminated from the data reported here. The most reliable data will relate to data acquisition systems (22 systems polled), report generators (22 systems polled), online systems (17 systems polled) and batch (8 systems polled). Only 3 process control systems were polled, and there were no significant correlations shown there. Attribute six shows that the majority of systems were classified to some degree with batch and report systems having the strictest degree of classification. All systems had a high degree of hardware interfacing, attribute nine. Data acquisition systems and online systems had a high degree of timing constraints while batch programs had a significantly low degree of

Table 1. Numbered Attributes

Number	Attribute
1	TYPE1
2	TYPE2
3	TIMING_CONSTRAINTS
4	MEMORY_CONSTRAINTS
5	LANGUAGE
6	SECURITY
7	NUMBER_OF_MODULES
8	COMPLEXITY
9	USE_OF_HARDWARE
10	LIBRARY
11	CONSULTANT
12	MANUALS
13	OBJECTIVES
14	INTERFACING
15	ANALYSIS_EXPERIENCE
16	DESIGN_EXPERIENCE
17	CODE_EXPERIENCE
18	NUMBER_ANALYSIS
19	NUMBER_DESIGN
20	NUMBER_CODE
21	PERCENT_ANALYSIS
22	PERCENT_DESIGN
23	PERCENT_CODE
24	PERCENT_DEBUG
25	STRUCTURED_ANALYSIS
26	STRUCTURED_DESIGN
27	STRUCTURED_CODE
28	MODULARITY
29	COHESION
30	blank
31	FAMILIARITY
32	EXISTING_DOCUMENTATION
33	UPDATING_DOCUMENTATION
34	HARDWARE_AFFECT
35	ANALYSIS_EXPERIENCE
36	DESIGN_EXPERIENCE
37	CODE_EXPERIENCE
38	FLEXIBILITY
39	DEVELOPMENT_SYSTEM
40	COMPLEXITY
41	EXTENT

Table 2. Correspondence Ratios for Development Attributes

Att.	Proc. Cont.	Data Acq.	Online	Graphics	Batch	Report
1	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
2	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
3	0.100000E+01	0.650000E+00	0.750000E+00	0.500000E+00	0.333333E+00	0.500000E+00
4	0.666667E+00	0.450000E+00	0.500000E+00	0.500000E+00	0.333333E+00	0.277778E+00
5	0.000000E+00	0.500000E-01	0.833333E-01	0.000000E+00	0.166667E+00	0.555556E-01
6	0.333333E+00	0.500000E+00	0.416667E+00	0.100000E+01	0.666667E+00	0.777778E+00
7	0.000000E+00	0.150000E+00	0.000000E+00	0.000000E+00	0.166667E+00	0.277778E+00
8	0.333333E+00	0.150000E+00	0.250000E+00	0.000000E+00	0.166667E+00	0.555556E-01
9	0.666667E+00	0.400000E+00	0.333333E+00	0.100000E+01	0.500000E+00	0.611111E+00
10	0.666667E+00	0.450000E+00	0.500000E+00	0.500000E+00	0.333333E+00	0.444444E+00
11	0.000000E+00	0.500000E+00	0.583333E+00	0.100000E+01	0.500000E+00	0.444444E+00
12	0.333333E+00	0.350000E+00	0.333333E+00	0.500000E+00	0.166667E+00	0.333333E+00
13	0.333333E+00	0.400000E+00	0.416667E+00	0.000000E+00	0.500000E+00	0.611111E+00
14	0.666667E+00	0.400000E+00	0.333333E+00	0.000000E+00	0.500000E+00	0.333333E+00
15	0.000000E+00	0.100000E+00	0.166667E+00	0.100000E+01	0.166667E+00	0.111111E+00
16	0.000000E+00	0.100000E+00	0.166667E+00	0.100000E+01	0.166667E+00	0.111111E+00
17	0.000000E+00	0.500000E-01	0.833333E-01	0.100000E+01	0.333333E+00	0.166667E+00
18	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
19	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
20	0.333333E+00	0.500000E-01	0.000000E+00	0.000000E+00	0.166667E+00	0.555556E-01
21	0.333333E+00	0.100000E+00	0.833333E-01	0.000000E+00	0.000000E+00	0.000000E+00
22	0.000000E+00	0.500000E-01	0.833333E-01	0.000000E+00	0.166667E+00	0.000000E+00
23	0.000000E+00	0.100000E+00	0.166667E+00	0.100000E+01	0.500000E+00	0.388889E+00
24	0.666667E+00	0.500000E+00	0.416667E+00	0.000000E+00	0.000000E+00	0.222222E+00
25	0.000000E+00	0.200000E+00	0.416667E+00	0.500000E+00	0.333333E+00	0.388889E+00
26	0.000000E+00	0.200000E+00	0.416667E+00	0.500000E+00	0.333333E+00	0.388889E+00
27	0.000000E+00	0.100000E+00	0.833333E-01	0.500000E+00	0.166667E+00	0.166667E+00
28	0.000000E+00	0.000000E+00	0.000000E+00	0.500000E+00	0.333333E+00	0.166667E+00
29	0.000000E+00	0.500000E-01	0.833333E-01	0.500000E+00	0.000000E+00	0.555556E-01

timing constraints. This would be expected considering the function of these types of systems.

There was a correlation between the application and the language used in that over 75 percent of all report generators and batch programs were done in Basic. Batch systems were the only type system to interface greatly with existing systems.

None of the systems had significantly high or low degrees of structure in any phase, although the degree of structured analysis and structured design used appeared to be similar.

The next report, Table 3, shows the third degree polynomial coefficients for each attribute. The least square fit equations yielded some interesting results. The weighted least squares approximation by polynomials was used [8] for both a second and third degree fit, for comparison. The third degree fits yielded more close fits and better fits by those attributes already well fit to a second degree polynomial. The weighted (by the number of projects having the attribute with a particular value or in a particular range) attribute values were used as the data points, versus the average normalized man-hour value for that attribute value, according to equation (4.1).

The method used selects the best fit. If a second degree polynomial fits better, it will be used and the smaller the x cubed coefficient, the closer the fit is to a second degree equation. This can be seen from the equations for attributes three and four which, as yes or no questions, obviously fit best to straight lines, as the method calculated.

Table 3. Development Conversion Coefficients
where $A + Bx + Cx^2 + Dx^3$ = manhours per line of source code
and x is the attribute value 1 through 6

Att.	A	B	C	D
1	-.207673E+01	-.391240E+01	0.728207E+00	-.378222E-01
2	-.592147E+00	-.590733E-01	0.542940E-01	-.104661E-01
3	0.232076E+01	-.952173E+00	0.000000E+00	0.000000E+00
4	-.416982E+00	0.896458E+00	0.000000E+00	0.000000E+00
5	0.132080E+01	-.370216E+01	0.103566E+01	-.100604E+00
6	0.377098E+01	0.115953E+01	-.530704E+00	0.549291E-01
7	0.976519E+00	-.206216E+01	0.434611E+00	-.290607E-01
8	0.878002E+01	0.991121E+01	-.343436E+01	0.372490E+00
9	0.104435E+02	0.191025E+01	-.847789E+00	0.109471E+00
10	0.173682E+01	0.390080E+01	-.840777E+00	0.549861E-01
11	0.657659E+00	0.277288E+01	-.605181E+00	0.372833E-01
12	0.337002E+01	0.342917E+01	-.918227E+00	0.818086E-01
13	-.584429E+01	-.247575E+01	0.934564E+00	-.977080E-01
14	0.189067E+02	0.116743E+02	-.370944E+01	0.347955E+00
15	-.132016E+01	-.494067E+01	0.138612E+01	-.122083E+00
16	-.207936E+00	-.350008E+01	0.842356E+00	-.625617E-01
17	0.110394E+01	-.316591E+01	0.704618E+00	-.488426E-01
18	-.243981E+01	-.983268E+01	0.478398E+01	-.664673E+00
19	-.638442E+01	-.544932E+01	0.355455E+01	-.707169E+00
20	0.362583E+01	-.171030E+00	-.263537E+00	0.523872E-01
21	-.101480E+02	-.817835E+01	0.332919E+01	-.405229E+00
22	-.470778E+01	-.210585E+01	0.134247E+01	-.212510E+00
23	0.675534E+01	0.241149E+01	-.912549E+00	0.919481E-01
24	-.405481E+00	0.541375E+00	-.621305E-01	-.642718E-02
25	-.157203E+02	-.612546E+01	0.249241E+01	-.271965E+00
26	-.118485E+02	-.244252E+01	0.136149E+01	-.174809E+00
27	-.622230E+01	-.180136E+01	0.987800E+00	-.132015E+00
28	0.796635E+01	0.141724E+01	-.897058E+00	0.123981E+00
29	0.357755E+01	0.642115E+01	-.235720E+01	0.283800E+00

It is not assumed here that all data fits best to any polynomial but rather that for a small range, (one to six) a polynomial equation would be an adequate function to use to model the data's behavior. The greater the degree of polynomial allowed, the more accurate a fit could be calculated. Due to limitations of the computer and time, a maximum of a third degree polynomial was deemed adequate for this application.

Table 4 gives the errors for the conversion equations when attribute values of one through six are plugged back into the equations and compared with the experimental value used to develop the equations. For instance, when a value of one was used for x for attribute one, system type, in the equation for attribute one, taken from Table 3,

$$-2.07673 - 3.9124x + 7.28207x^2 - .0378222x^3 = y$$

where y is man-hours per line of source code, the value of 1.2551178 differs from the value used to calculate the equation by -.507571E-07.

In general then, the numbers in Table 4 indicate the accuracy or validity of the conversion equation. However, if there was no data for the upper ranges, a zero appears in the table. For example, attribute 4 can only have a yes or no answer, so it has only two entries in the table. Attribute three, on the other hand, has zero error in the conversion equation because, we know from Table 3 that it fits a linear equation with values of one and two. Three is the only attribute with zero error everywhere.

The attributes which fit a curve best, as observed by having the lowest errors were the type, timing and memory constraints, percentage of design time, the number of people participating in the analysis and design phases of a project, the module size and the cohesion with

Table 4. Development Conversion Equation Errors

Att.	1	2	3	4	5	6
1	-.507571E-07	-.521541E-07	0.000000E+00	0.265427E-07	0.000000E+00	0.000000E+00
2	-.786438E-02	0.196611E-01	-.235933E+00	0.589831E-01	-.117968E-01	0.000000E+00
3	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
4	0.119209E-06	0.298023E-07	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
5	0.333112E-03	-.106608E-02	0.119922E-01	-.159899E-01	0.399762E-02	0.000000E+00
6	-.123048E-01	0.131250E+00	-.295314E+00	0.393751E+00	-.579052E-02	0.000000E+00
7	0.248738E-02	-.665065E-01	0.136167E+00	0.261570E-01	-.769692E-01	0.649296E-01
8	-.625256E-01	0.909463E-01	-.136420E+00	0.250102E+00	-.250102E+00	0.000000E+00
9	0.270026E+00	-.215836E+00	-.110223E-01	0.757729E+00	-.162261E+00	0.865187E-01
10	-.544448E-01	0.118788E+00	-.980004E+00	0.435557E+00	-.362964E-01	0.000000E+00
11	0.416990E+00	-.649232E+00	0.120696E+01	-.133855E+01	0.102396E+01	-.799247E-01
12	0.267588E+00	-.451476E+00	0.467966E+00	-.659599E-01	-.804012E+00	0.167398E+00
13	-.242418E+00	0.563776E+00	-.281943E+00	-.131530E+01	0.664392E+00	-.293142E+00
14	-.368184E+00	0.208779E+01	-.949184E+00	0.380907E+00	0.128016E+01	-.122257E+00
15	0.322505E-01	-.120940E+00	0.181410E+00	-.241879E+00	0.725641E-01	0.000000E+00
16	-.186682E-01	0.672057E-01	-.140012E+00	0.124455E+00	-.933427E-02	0.000000E+00
17	0.249664E-01	-.742237E-01	0.172720E+00	-.237287E+00	0.125901E-01	0.174333E-01
18	0.131602E-08	0.981530E-08	0.199870E-07	0.000000E+00	0.000000E+00	0.000000E+00
19	-.338988E-07	-.170087E-06	0.144848E-07	0.000000E+00	0.000000E+00	0.000000E+00
20	-.109207E-02	0.742560E-02	-.556919E-01	0.742560E-01	-.185639E-01	0.000000E+00
21	0.470382E-01	-.136838E+00	0.225783E+00	-.150522E+01	0.376305E+00	0.000000E+00
22	-.134110E-06	-.596046E-07	-.596046E-07	-.894070E-07	0.000000E+00	0.000000E+00
23	-.396577E+00	0.247860E+00	-.440641E+00	0.283269E+00	-.198289E+00	0.000000E+00
24	0.525701E+00	-.820005E+00	0.653557E+00	-.483029E+00	-.348194E+00	0.284186E+00
25	-.215325E+00	0.391501E+00	-.143550E+01	0.430650E+00	-.172261E+00	0.000000E+00
26	0.145558E+00	-.166649E+00	-.880778E+00	0.518961E+01	-.172353E+01	0.803047E+00
27	0.206882E+00	-.545689E+00	0.126227E+01	-.205076E+01	-.868031E+00	0.899500E+00
28	-.119209E-06	-.894070E-07	-.126660E-06	-.208616E-06	0.000000E+00	0.000000E+00
29	-.894070E-07	0.372529E-07	0.894070E-07	-.149012E-06	0.000000E+00	0.000000E+00

maximum errors from Table 4 of: -.52E-7, 0.0, .12E-6, .2E-7, -.17E-6, -.13E-6, -.21E-6, and -.15E-6, respectively. The conversion equations from Table 3 are:

TYPE	$-2.08 - 3.91x + .73x^2 - .04x^3 = y$	(5.1)
TIMING CONSTRAINTS	$2.32 - .95x = y$	(5.2)
MEMORY CONSTRAINTS	$-4.2 + .90x = y$	(5.3)
NUMBER ANALYSIS	$-2.44 - 9.83x + 4.78x^2 - .66x^3 = y$	(5.4)
NUMBER DESIGN	$-6.38 - 5.45x + 3.55x^2 - .71x^3 = y$	(5.5)
PERCENT DESIGN	$-4.71 - 2.11x + 1.34x^2 - .21x^3 = y$	(5.6)
MODULE SIZE	$7.0 + 1.42x - .90x^2 .12x^3 = y$	(5.7)
COHESION	$3.58 + 6.42x - 2.36x^2 + .28x^3 = y$	(5.8)

The percentage of time spent on design was remarkably fit to a curve (maximum error of -.13E-6) in comparison to the percentages of time spent on other development phases (maximum errors of: -1.5 for analysis, -.44 for coding, and -.82 for debugging). This may be attributable to the ease of distinguishing between the design phase and the coding phase. There is difficulty in separating analysis and design, especially if a structured methodology is not used. And certainly it is difficult to separate coding from debugging and testing.

An attribute which did not fit well was complexity, where the high degree of complexity fit best, but every task can be thought of as being complex to some degree. Plus, outside influences such as management may add complexity to a task. Hardware interfacing did not fit well, but as seen earlier, all systems rated this attribute high. Availability of libraries and consultants, adequacy of manuals and objectives, entropy or interfacing with existing software, and the experience level of the

software personnel all did not fit well to a curve. With respect to experience, often the more experienced software person is given the more complex and larger projects. This would counter the assumption that they would be able to do a job faster. Also, the quality of the software product may come into play, and that is a factor which was not considered here.

The modularity, the number of lines of code per module, fit well as did the cohesion measure. These results would indicate that these attributes can be used in this estimating algorithm with more reliability than some of the others. So, a run using only the more accurately fitting attributes was done.

The results showed a very high standard deviation. This was before the correspondence ratios were incorporated into the algorithm. However, using this data and data gathered by inspection from reports generated for each attribute, it became obvious that several data sets for projects were totally out-of-line and it was determined that these data sets should be eliminated from the system and all analysis programs should be rerun.

Table 5 shows the results of this run, the estimated development time, the error for each project, the percentage of error for each project, and the standard deviation of the percentages of error. The lowest percent error was 25% for PROJ15 and the highest percent error was 1900%, PROJ6. The standard deviation for the percentage of error was 592%.

The percentages were high, high enough to introduce some doubt as to the possibility of using the algorithm for general applications.

Table 5. Development Estimates

Proj.	Act. Time	Est. Time	Difference	Per. Err.
PROJ1	0.900000E+04	0.712753E+05	0.622753E+05	691.948
PROJ10	0.150000E+04	0.835000E+02	-.141650E+04	94.4333
PROJ11	0.100000E+04	0.200400E+03	-.799600E+03	79.9600
PROJ13	0.200000E+04	0.597455E+02	-.194025E+04	97.0127
PROJ15	0.270000E+05	0.200400E+05	-.696000E+04	25.7778
PROJ16	0.330000E+04	0.106612E+04	-.223388E+04	67.6934
PROJ17	0.200000E+04	0.835000E+03	-.116500E+04	58.2500
PROJ18	0.300000E+04	0.167000E+04	-.133000E+04	44.3333
PROJ19	0.250000E+04	0.410311E+04	0.160311E+04	64.1246
PROJ2	0.250000E+04	0.529941E+04	0.279941E+04	111.976
PROJ20	0.200000E+04	0.149305E+03	-.185070E+04	92.5348
PROJ21	0.300000E+04	0.133325E+05	0.103325E+05	344.417
PROJ22	0.480000E+03	0.238576E+03	-.241424E+03	50.2966
PROJ23	0.400000E+04	0.227080E+04	-.172920E+04	43.2300
PROJ24	0.330000E+04	0.720091E+03	-.257991E+04	78.1791
PROJ25	0.350000E+05	0.133985E+05	-.216015E+05	61.7185
PROJ27	0.350000E+03	0.200400E+03	-.149600E+03	42.7429
PROJ29	0.100000E+04	0.190928E+02	-.980907E+03	98.0907
PROJ3	0.400000E+04	0.300600E+05	0.260600E+05	651.500
PROJ30	0.750000E+04	0.825519E+02	-.741745E+04	98.8993
PROJ31	0.100000E+04	0.159432E+04	0.594322E+03	59.4322
PROJ33	0.250000E+03	0.163923E+03	-.860775E+02	34.4310
PROJ34	0.150000E+04	0.239386E+03	-.126061E+04	84.0409
PROJ35	0.100000E+04	0.422451E+03	-.577549E+03	57.7549
PROJ37	0.400000E+04	0.792174E+03	-.320783E+04	80.1956
PROJ38	0.600000E+04	0.972716E+03	-.502728E+04	83.7881
PROJ39	0.220000E+04	0.125250E+04	-.947500E+03	43.0682
PROJ4	0.100000E+04	0.100200E+05	0.902000E+04	902.000

Table 5 (continued)

Proj.	Act. Time	Est. Time	Difference	Per. Err.
PROJ5	0.300000E+04	0.263321E+05	0.233321E+05	777.735
PROJ6	0.200000E+04	0.400641E+05	0.380641E+05	1903.20
PROJ9	0.200000E+04	0.801600E+02	-.191984E+04	95.9920

Standard Deviation of the percentage of error = 391.966

However, many of the results did fall in the range mentioned by Putnam [30], where management can influence productivity by plus or minus 25%. Considering, also, the allowance of up to 28 times higher productivity for a high level performer [11]—most of the systems surveyed did fall in a size range allowing for a single programmer to implement the system—these results do fall in ranges found valid by others.

A similar process to calculating development estimates was developed for calculating modification estimates. Table 6 shows the coefficients calculated for the twenty-nine system attributes, also used for the development estimate, and the added eleven attributes pertaining only to modifications. Number thirty, as seen on Table 1, was left blank to allow for further expansion of the development attribute set. Therefore, all entries corresponding with the thirtieth attribute will be zero. Table 7 shows the errors for the conversion equations for modifications just as Table 4 does for development. From this table it can be seen that the type of system, the timing and memory constraints, number of people on the job, especially in analysis and design, the size the modules and the cohesion are most accurately fit to a curve with maximum errors from Table 7 of: .24E-6, -.53E-14, 0.0, -.18E-7, .12E-6, -.98E-7, -.12E-6, .28E-6, and .10E-6, respectively. The best fitting factor of the new attributes was the type of change being made with a maximum error of -.31E-6. Their conversion equations from Table 6 are:

TYPE $-9.78 - 11.20x + 2.07x2 - .12x3 = y$ (5.9)

TIMING CONSTRAINTS $-11.84 - 26.59x + 18.x2 - 4.x3 = y$ (5.10)

MEMORY CONSTRAINTS $28.4 - 2.76x + 1.x2 = y$ (5.11)

NUMBER ANALYSIS $-1.98 - 4.36x + 2.29x2 - .39x3 = y$ (5.12)

Table 6. Modification Conversion Coefficients
where $A + Bx + Cx^2 + Dx^3$ = manhours per source line of code
and x is the attribute value 1 through 6

Att.	A	B	C	D
1	-.977945E+01	-.111960E+02	0.207084E+01	-.122466E+00
2	0.133335E+02	0.863112E+01	-.191886E+01	0.134935E+00
3	-.118384E+02	-.265897E+02	0.180000E+02	-.400000E+01
4	0.284033E+01	-.276007E+01	0.100000E+01	0.000000E+00
5	0.772226E+00	-.981958E-01	0.340596E-01	-.985503E-02
6	-.240720E+01	-.388581E+01	0.134141E+01	-.138959E+00
7	0.664591E+00	-.118312E+01	0.280020E+00	-.193517E-01
8	0.416744E+01	0.406422E+01	-.162062E+01	0.198725E+00
9	-.195661E+00	-.158736E+00	0.924102E-01	-.130244E-01
10	-.963800E+00	-.217212E+01	0.529942E+00	-.405528E-01
11	0.616773E+01	0.161579E+01	-.584809E+00	0.639408E-01
12	-.647907E+01	-.374365E+01	0.128948E+01	-.130416E+00
13	-.112274E+01	-.112999E+01	0.365854E+00	-.356636E-01
14	0.175008E+01	-.602880E+00	0.998022E-01	-.168902E-02
15	0.641188E+01	0.360956E+01	-.125084E+01	0.135810E+00
16	0.491821E+01	0.360713E+01	-.101657E+01	0.885544E-01
17	-.659685E+00	-.196016E+01	0.545589E+00	-.468413E-01
18	-.197742E+01	-.435863E+01	0.228697E+01	-.388948E+00
19	0.267925E+01	0.130101E+01	-.916219E+00	0.151703E+00
20	-.432750E+00	-.135961E+01	0.536318E+00	-.718021E-01
21	0.948963E-01	-.527203E+00	0.344491E+00	-.306917E-01
22	-.129926E+01	0.423315E+00	0.107819E+00	-.470097E-01
23	0.255986E+01	-.158286E+00	-.730893E-01	0.765037E-02
24	0.397988E+01	0.276749E+01	-.795158E+00	0.738323E-01
25	-.467999E+01	-.188372E+01	0.760074E+00	-.882105E-01

Table 6 (continued)

Att.	A	B	C	D
26	-.177788E+01	-.636809E+00	0.300339E+00	-.408576E-01
27	-.315426E+00	-.134459E+01	0.428842E+00	-.452415E-01
28	-.646816E+01	-.873703E+01	0.282783E+01	-.302839E+00
29	0.373659E+00	0.192027E+01	-.578293E+00	0.408052E-01
30	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
31	-.275218E-01	0.170636E+00	0.524064E-01	-.125647E-01
32	0.201875E+00	0.168165E+01	-.355746E+00	0.218183E-01
33	0.297930E+01	-.326174E+00	-.343869E-01	0.118615E-01
34	0.714954E+01	0.439726E+01	-.132466E+01	0.112556E+00
35	0.193268E+00	0.113687E+01	-.230091E+00	0.113961E-01
36	0.141895E+01	0.190366E+01	-.499114E+00	0.381603E-01
37	0.548137E+01	0.627272E+01	-.274149E+01	0.371848E+00
38	0.258775E+00	-.229356E+01	0.673103E+00	-.667967E-01
39	0.213174E+01	0.118891E+01	-.337478E+00	0.321389E-01
40	0.592147E+01	0.185429E+01	-.661615E+00	0.623058E-01
41	-.160160E+01	-.373480E+01	0.142213E+01	-.151753E+00

Table 7. Modification Conversion Equation Errors

Att.	1	2	3	4	5	6
1	-.344589E-07	0.238419E-06	0.208616E-06	-.130385E-07	0.000000E+00	0.000000E+00
2	-.159359E-01	0.289746E-01	-.318718E+00	0.796797E-01	-.141651E-01	0.000000E+00
3	-.532907E-14	-.355271E-14	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
4	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
5	0.145183E-01	-.425872E-01	0.479105E+00	-.638807E+00	0.159702E+00	0.000000E+00
6	0.121550E-01	-.113447E+00	0.510510E+00	-.340340E+00	0.472687E-02	0.000000E+00
7	0.189863E-01	-.476940E+00	0.805115E+00	0.573192E+00	-.706217E+00	0.551323E+00
8	0.447035E-07	-.596046E-07	-.149012E-07	0.298023E-07	0.000000E+00	0.000000E+00
9	-.473282E+00	0.669157E+00	-.564035E-01	-.717876E+00	0.218246E+00	-.110963E+00
10	-.264036E-01	0.633685E-01	-.475264E+00	0.211228E+00	-.176024E-01	0.000000E+00
11	-.248557E+00	0.919915E-01	0.462956E-01	-.414861E+00	0.506853E+00	-.552847E-01
12	-.204122E+00	0.191170E+00	0.689745E-03	-.102371E+01	0.153402E+01	-.230025E+00
13	-.121313E+00	0.304959E+00	-.837629E-02	-.994192E+00	0.428476E+00	-.150246E+00
14	-.378591E-01	0.143082E+00	0.251384E-01	-.605216E+00	0.561224E+00	-.409920E-01
15	-.533349E-01	0.160004E+00	-.304770E+00	0.355565E+00	-.106670E+00	0.000000E+00
16	-.209590E-01	0.617337E-01	-.169768E+00	0.125754E+00	-.943175E-02	0.000000E+00
17	-.172165E-01	0.254804E-02	0.186213E+00	-.637698E+00	0.775430E+00	-.323945E+00
18	-.176162E-07	0.161482E-07	-.149012E-07	0.000000E+00	0.000000E+00	0.000000E+00
19	0.169666E-07	0.324694E-07	0.120572E-06	0.000000E+00	0.000000E+00	0.000000E+00
20	0.820720E-02	-.525261E-01	0.393946E+00	-.525261E+00	0.131315E+00	0.000000E+00
21	-.982545E-07	-.763685E-07	-.596046E-07	-.521541E-07	0.000000E+00	0.000000E+00
22	-.121072E-06	-.707805E-07	-.298023E-07	0.745058E-08	0.000000E+00	0.000000E+00
23	0.540184E+00	-.184154E+00	0.270092E+00	-.192923E+00	0.135046E+00	0.000000E+00
24	0.106007E+00	-.133333E+00	0.146654E+00	-.271946E+00	0.184377E+00	-.426396E-01

Table 7 (continued)

Att.	1	2	3	4	5	6
25	0.201590E+00	-.335984E+00	0.111995E+01	-.279987E+00	0.134393E+00	0.000000E+00
26	0.143387E+00	-.313794E+00	0.673334E+00	0.130807E+00	-.268047E+00	0.205717E+00
27	0.163611E-01	-.652452E-01	0.189547E+00	-.969670E+00	0.133676E+00	-.634440E-01
28	0.596046E-07	-.298023E-07	0.447035E-07	0.283122E-06	0.000000E+00	0.000000E+00
29	0.100583E-06	0.372529E-08	-.372529E-08	0.819564E-07	0.000000E+00	0.000000E+00
30	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
31	0.337932E-01	-.289656E+00	0.101379E+01	-.506897E+00	0.405518E+00	0.000000E+00
32	0.358415E-01	-.238608E+00	0.103696E+00	0.363466E+00	-.114172E+01	0.127998E+00
33	0.164252E-01	-.119686E+00	0.541038E-01	0.185269E+00	-.265059E+00	0.406607E-01
34	0.373810E-01	-.186907E+00	0.186906E+00	-.373813E+00	0.467264E-02	0.000000E+00
35	-.153386E-01	0.126593E-01	0.835742E-01	-.769870E+00	0.120544E+01	-.820506E-01
36	-.506907E-01	0.886862E-01	-.884641E-01	-.621693E+00	0.142031E+01	-.887576E-01
37	-.182539E-06	-.238419E-06	-.596046E-07	0.357628E-06	0.000000E+00	0.000000E+00
38	0.251870E-01	-.402991E-01	0.503738E-01	-.201496E+00	0.503739E-01	0.000000E+00
39	-.453086E-01	0.597261E-01	0.140011E-01	-.279048E+00	0.523092E+00	-.102285E+00
40	-.221788E+00	0.368651E+00	-.208953E+00	-.477257E-02	0.125867E+00	-.499488E-01
41	-.216067E-06	-.312924E-06	-.275671E-06	-.782311E-07	0.000000E+00	0.000000E+00

NUMBER DESIGN	$2.68 + 1.30x - .92x^2 + .15x^3 = y$	(5.13)
PERCENT ANALYSIS	$.09 - .53x + .34x^2 - .03x^3 = y$	(5.14)
PERCENT DESIGN	$-1.3 + .42x + .11x^2 - .05x^3 = y$	(5.15)
MODULE SIZE	$-6.47 - 8.74x + 2.83x^2 - .30x^2 = y$	(5.16)
COHESION	$.37 + 1.92x - .58x^2 + .04x^3 = y$	(5.17)
EXTENT	$-1.6 - 3.73x + 1.42x^2 - .15x^3 = y$	(5.18)

In general, though all modification factors were modeled well by a polynomial curve, only those development factors which appeared to be poor indicators or logically not applicable were disregarded for further testing. Then, using just these factors, the modification estimate was generated.

It is interesting to notice that the structuredness of the development phase, both analysis and design structure had the highest error in the curve fitting, indicating that they may have a random distribution and little influence over productivity. Yet cohesion and modularity factors fit very well to polynomial curves, indicating that their influence over an estimate using this method would be more reliable. The quality of these attributes should directly relate to the analysis and design phases.

Also, the effect of updating documentation was found to have a more predictable influence over productivity than was documentation already existing. And, updating documentation lowered the amount of time spent on a change. Maybe this is because documentation could only be updated if it existed in the first place.

The complexity of the modification seemed relatively unimportant, in contrast to the complexity in development systems which is directly

proportional to the size of a system which in turn is directly proportional to the effort. Modifications are generally small, and "small" is generally perceived as being less complicated.

Table 8 shows the results of running the modification estimate program. The lowest percentage error is 25%, PROJ37, the highest is 3000%, PROJ9. Most errors were in the 50 to 100 percent range. The standard deviation for the percentage of error was 565%, not much better than that for the development system estimates.

Since the algorithm did not provide a solution to the estimating problem, it was decided to look at the trends from the information in the various tables. By evaluating the conversion equations at their various values, one through six, in most cases, the shape of the curve should indicate the affect the attribute has on productivity, that is, man-hours per line of source code.

First, consider the development conversion equations, and look only at those with the lower errors from Table 3. The type of system shows a nearly horizontal line. Timing constraints decrease the productivity while memory constraints increase productivity. This may be because a limited memory also limits the size and complexity of a system. Programs coded in Fortran had the lowest productivity rate, those in Basic had the highest. The order from lowest to highest productivity of languages was Fortran, Cobol, Pascal, PL1, and Basic. Except for Basic, the more structured the language, the higher the productivity. Basic may lead in highest productivity because of its limited application to smaller systems. It simply cannot be used for large, complex, interactive systems, the language itself is too restrictive.

Table 8. Modification Estimates

Proj.	Act. Time	Est. Time	Difference	Per. Err.
PROJ1	0.300000E+01	0.835000E+01	0.535000E+01	178.333
PROJ10	0.400000E+02	0.835000E+01	-.316500E+02	79.1250
PROJ11	0.200000E+03	0.334000E+02	-.166600E+03	83.3000
PROJ13	0.900000E+02	0.200400E+02	-.699600E+02	77.7333
PROJ15	0.350000E+03	0.250500E+02	-.324950E+03	92.8429
PROJ16	0.100000E+04	0.417500E+02	-.958250E+03	95.8250
PROJ17	0.200000E+02	0.835000E+01	-.116500E+02	58.2500
PROJ19	0.400000E+03	0.417500E+02	-.358250E+03	89.5625
PROJ2	0.240000E+02	0.000000E+00	-.240000E+02	100.000
PROJ20	0.100000E+03	0.835000E+01	-.916500E+02	91.6500
PROJ21	0.200000E+01	0.835000E+01	0.635000E+01	317.500
PROJ22	0.800000E+01	0.133600E+02	0.536000E+01	67.0000
PROJ23	0.400000E+03	0.167000E+02	-.383300E+03	95.8250
PROJ24	0.400000E+02	0.835000E+01	-.316500E+02	79.1250
PROJ25	0.200000E+02	0.835000E+01	-.116500E+02	58.2500
PROJ27	0.160000E+02	0.835000E+01	-.765000E+01	47.8125
PROJ29	0.100000E+03	0.167000E+02	-.833000E+02	83.3000
PROJ3	0.180000E+03	0.501000E+02	-.129900E+03	72.1667
PROJ30	0.175000E+03	0.835000E+01	-.166650E+03	95.2286
PROJ31	0.700000E+03	0.167000E+02	-.683300E+03	97.6143
PROJ32	0.170000E+03	0.835000E+01	-.161650E+03	95.0882
PROJ33	0.500000E+02	0.000000E+00	-.500000E+02	100.000
PROJ34	0.175000E+03	0.835000E+01	-.166650E+03	95.2286
PROJ37	0.200000E+02	0.250500E+02	0.505000E+01	25.2500
PROJ38	0.400000E+03	0.501000E+02	-.349900E+03	87.4750
PROJ39	0.400000E+03	0.835000E+02	-.316500E+03	79.1250
PROJ4	0.100000E+03	0.250500E+02	-.749500E+02	74.9500

Table 8 (continued)

Proj.	Act. Time	Est. Time	Difference	Per. Err.
PROJ5	0.520000E+03	0.417500E+02	-.478250E+03	91.9712
PROJ6	0.540000E+03	0.417500E+02	-.498250E+03	92.2685
PROJ9	0.170000E+03	0.557563E+04	0.540563E+04	3179.78

Standard Deviation of the percentage of error = 565.654

The greater the security restrictions, the lower the productivity. The number of people participating in analysis along with the percentage of time spent on design, have the best productivity at their extremes. One possible explanation is that it is best to assign a single person to a job if feasible. If this is not feasible, organize a formal project team, to deal effectively with the communication factor that is introduced in group work. In design, doing a token design has a worse effect on productivity than does doing no design at all. With respect to modularity, as we would expect, there is a module size range of 50 to 100 lines of code per module which provides the best productivity rate. The cohesion shows that the better the cohesion rating, the better the productivity.

Analyzing the modification conversion equations, we find that the system type, timing constraints, memory constraints, and number of people participating in the analysis have similar effects on modification productivity, as on development productivity. However, the number of people working on design showed that as the number of people increased, the productivity decreased. This may indicate a lack of standards to unify a system and make it easier to understand for the maintenance programmer. It also reinforces the significance of communication on productivity.

In observing the percentage of time spent on analysis in the development phase, it is interesting to note that it seems to fit much better to a curve in the modification phase and indicates that as the percentage of time spent on analysis increases, the productivity decreases. At first glance, this would seem inconsistent with previous

research except that the modifications made to a system developed with thorough analysis would be major enhancements, and would be more complex.

The percentage of time devoted to design had the opposite effect: as design time increased, productivity increased. Then jumping to the modularity and cohesion measures, which should be a result of thorough design, we have consistent data indicating that cohesion and modularity each have a positive affect on productivity, agreeing with the results for the development estimates and verifying the importance of thorough design on software life cycle costs.

The two most reliable attributes for modification estimating, according to the errors in Table 7, are the coding experience of the programmer and the extent of the modification. Once again with respect to experience, it would seem that the more difficult jobs are given to the more experienced programmers. The extent of the change indicated that deleting modules or adding lines lowered productivity more than changing logic, adding modules and deleting lines lowered productivity.

CHAPTER VI

CONCLUSIONS

The cost estimating system contains most, if not all of the significant attributes needed to predict the cost of a system, either initial development or subsequent modifications. Many of the attributes fit very well to a polynomial of third degree or less, suggesting that polynomials can be used in the estimation algorithm. However, several attributes did not fit. This may mean that their distribution is random over the sampling, or they fit some function other than a polynomial, or they fit a higher degree polynomial. Further research could determine some of these by simply increasing the maximum degree of the polynomial allowed.

From the coefficients found, some attributes fit linear equations and some fit more nearly to a second order polynomial than to a third order polynomial, positively reinforcing the choice of polynomial curve fitting for the attribute weights. The expansion of the value responses from the standard three to six allowed for producing this type of relationship.

The estimating algorithm itself would not be adequate, as it is, in providing accurate cost estimates for development systems due to the large errors. The range of systems, both by type and by size, used for data was very broad, no controls were used at all. No weight was given to any attribute to allow for that attribute to have more influence over

an estimate. Due to the increased accuracy of the estimates when the correspondence ratios were incorporated into the estimating system it would seem highly likely that weighting factors might significantly increase the accuracy of the model.

Although the number of lines in the system was not originally intended to be used in the estimating algorithm, it was indirectly incorporated by using the estimated number of modules with the estimated number of lines per module. Another variable which should have been used in conjunction with these two or possibly instead of the number of modules would have been the number of reports or files required. This serves as a good estimator for business applications, though it is rather insignificant with respect to real-time applications.

Despite being deliberately vague in the survey regarding the purpose of this research, to avoid people answering questions with respect to the way they thought a system should be developed, rather than the way it was developed, some results are contradictory in regard to questions addressing the structure of the system. This is clearly seen by the discrepencies in the modification data and alluded to in the systems data. The increased attention to the use of structured methodologies may have influenced the answers to these questions, and it would seem logical to assume that answers to simple quantitative questions, for instance "average length of a module", would be less biased. Hence, the accuracy of the curve fitting for this attribute can be assumed more reliable, and if the equation is checked, it can be seen that there is an average module size that is best, but that increasing module size increases cost much more dramatically than decreasing size.

Another puzzling result of the survey was that existing documentation did not have a significant effect on modification, yet updating the documentation tended to have a decreasing affect on time needed to make a change. But documentation can only be updated if it exists, which would imply that having existing up-to-date documentation does make a positive difference.

In summary, the following conclusions have been made:

1. The survey was somewhat biased in its terminology and may have been somewhat incomplete in its content. The result of this would be an incomplete database with some inaccurate data.
2. Some data fit very well to a polynomial curve and therefore those attributes may be considered valid and the polynomial curve may be considered an adequate function for calculating attribute value weights.
3. The data that did not fit the polynomial curve may fit a more complex function and therefore should not be disregarded totally.
4. Correspondence ratios distributing the weight of one attribute with respect to another are significant and yield a more accurate estimate.
5. No weights were given to any attribute for importance and this probably contributed to the high error of the system.
6. The introduction of multiple standards had a dispersing affect on the data. Controls should be used to limit the standards of comparison.
7. No measure was done or even exists for the quality of software.

When one is developed, it could be used to greatly enhance this system and increase the information obtained from it.

The existing data base provides enough information for an estimating model to be developed given consistent data which would be obtained by controlling the data to reflect only systems of a single unified group, using one set of standards.

LIST OF REFERENCES

LIST OF REFERENCES

1. Baker,F.T., 'Chief Programmer Team Management of Production Programming,' IBM Systems Journal, Vol. 11, no. 1,1972, pp. 56-73.

2. Basili, V. and Zelkowitz, M., 'Analyzing Medium Scale Software Development,' 3rd International Conference on Software Engineering, Atlanta, GA, May 1978, pp. 116-123.

3. Beck, L.L. and Perkins, T.E., 'A Survey of Software Engineering Practice: Tools, Methods, and Results,' IEEE Transactions on Software Engineering, September 1983, p. 541.

4. Boehm, Barry W., Software Engineering Economics, Prentice-Hall, Inc., 1981. (800)526-0485

5. Brooks, F.P., Jr., The Mythical Man-Month, Reading, MA, Addison- Wesley, 1975.

6. Cave, W.C. and Salisbury, A.B., 'Controlling the Software Life Cycle-- The Project Management Task,' IEEE Transactions on Software Engineering, July 1978, p. 326.

7. Chapin, Ned, 'A measure of Software Complexity,' AFIPS Conference Proceedings, Vol 48, 1979, p. 995.

8. Collofello, J.S. and Yau, S.S., 'Some Stability Measures for Software Maintenance',IEEE Transactions of Software Engineering, Nov. 1980, p. 545.

9. Conte,S.D. and DeBoor, Carl, Elementary Numerical Analysis, McGraw- Hill, 1965, pp. 255-261.

10. Cooper, John D., 'Corporate Level Software Management', IEEE Transactions on Software Engineering, July 1978, p. 319.

11. Curtis, B. et. al, 'Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics,' IEEE Transactions on Software Engineering, March 1979, p. 96.

12. de Frietas, 'Method for Time Analysis of Programs,' IBM Systems Journal, Vol. 19, no. 4, p 414.

13. DeRoze, B.C., 'Defense Systems Software Management Plan,' Office of Assistant Secretary of Defense (Installations and Logistics), DDC Accession Number AD-A022 558, March 19,1976.

14. DeRoze, B.C. and Nyman, T.H., 'The Software Life Cycle - A Management and Technological Challenge in the Department of Defense,' IEEE Transactions on Software Engineering, July 1978, p. 309.

15. Dershowitz, N. and Manna, Z., 'The Evolution of Programs: Automatic Program Modification,' IEEE Transactions of Software Engineering, Nov. 1977, p. 377.

16. Feuer, A.R. and Fowlkes, E.B., 'Relating Computer Program Maintainability to Software Measures,' AFIPS Conference Proceedings, Vol.48, 1979, p. 1003.

17. Gehring, P.F. and Pooch, U.W., 'Software Development Management,' Data Management, Feb. 1977, pp. 14-18.

18. Gehring, P.F., 'Improving Software Development Estimates of Time and Cost,' 2nd International Conference on Software Engineering, San Francisco, CA, Oct. 13, 1976.

19. Goldberg, J., 'Proceedings of a Symposium of the High Cost of Software,' held at Naval Post Graduate School, Monterey, CA, DDC Accession Number AD-770 121, Sept. 17-19, 1973.

20. Kapur,Gopal, 'Software Maintenance,' Computer World, Vol. XVIII, No.39, September 26,1983, p.ID/13.

21. McCabe, T., 'A Complexity Measure', IEEE Transactions on Software Engineering, Dec. 1976, pp. 308-320.

22. McHenry, R.C. and Walston, C.E., 'Software Life Cycle Management: Weapon Process Developer,' IEEE Transactions of Software Engineering, July 1978, p 334.

23. Morin, L.H., 'Estimation of Resources for Computer Programming Projects,' M.S. thesis, University of North Carolina, Chapel Hill, NC, 1973.

24. Moriconi, M.S., 'A Designer/Verifier's Assistant,' IEEE Transactions on Software Engineering, July 1979, p. 387.

25. Norden, P.V., 'Useful Tools for Project Management,' Management of Production, M.K. Starr, Editor, Baltimore MD, Penguin, 1970, pp 71-101.

26. Parnas, D.L., 'Designing Software for Ease of Extension and Contraction,' IEEE Transactions on Software Engineering, March 1979, p. 128.

27. Peercy, D., Software Maintainability Evaluation Guidelines Handbook, BDM/TAC-78-687-TR, Dec. 1978.

28. Peercy, D. and Paschich, T., 'Software Maintainability Analysis Program User's Manual,' BDM/TAC-78-697-TR, Dec. 1978.

29. Peercy, D., 'A Software Maintainability Evaluation Methodology,' IEEE Transactions on Software Engineering, July 1981, p. 343.

30. Putnam, L.H., 'A General Empirical Solution to the Macro Software Sizing and Estimating Problem,' IEEE Transactions on Software Engineering, July 1978, p. 345.

31. Putnam, L.H. and Wolverton, R.W., 'Quantitative Management: Software Cost Estimating,' (tutorial), IEEE Computer Society, Nov. 1977.

32. Putnam, L.H., 'A Macro-Estimating Methodolgy for Software Development,' Digital Papers, Fall COMPCON '76, 13th IEEE Computer Society International Conference, Sept. 1976,pp. 138-143.

33. Putnam, L.H., 'Measurement Data to Support Sizing, Estimating and Control of the Software Life Cycle,' Proceedings, COMPCON '78, IEEE, New York, 1978.

34. Ragland, F. and Peercy, D., 'Analysis of Software Maintainability Evaluation Process,' BDM/TAC-78-698-TR, Dec. 1978.

35. Snyder, T.R., 'Rate Charting,' Datamation, pp 44-47, Nov 1976.

36. Thayer, P., 'Software Maintainability Evaluation Methodology,' Air Force Test and Evaluation Center Report, June 1978.

37. Walston, C.E. and Felix, C.P., 'A Method of Programming Measurement and Estimation,' IBM Systems Journal, Vol. 16, no. 1, 1977.

38. Warburton, R.D.H., 'Managing and Predicting the Costs of Real-Time Software,' IEEE Transactions on Software Engineering,September 1983, p. 562.

39. Wolverton, R.W., 'The Cost of Developing Large Scale Software,' IEEE Transactions on Computers, Vol. 23, no. 6, 1974, pp. 615-636.

40. Zelkowitz, Marvin V., Shaw, Alan C. and Gannon, John D., Principles of Software Engineering and Design, Prentice-Hall, Inc.

41. 'The Management of Software Engineering,' IBM Systems Journal, Vol. 19, no. 4, p 414.

42. 'Proceedings: Software Cost and Sizing Workshop,' USAF Electronic Systems Division, Hanscom Field, MA, Oct. 1974.

43. 'Life Cycle Events,' Software Acquisition Management Guide Book Series, USAF Electronic Systems Division, Hanscom Field, MA, Technical Report 77-22, DDC Accession Number AD-A037 115, Feb. 1977.

44. 'Software Development and Maintenance Facilities,' Software Acquisition Management Guidebook Series, USAF Electronic Systems Division, Hanscom AFB, MA, DDC Accession Number AD-A038 234, Apr 1977.

45. 'Estimating Software Resource Requirements,' Structured Programming Series, USAF Rome Air Development Center Griffis AFB, NY, Vol 1-15, DDC Accession No. AD-A016 416, July 1975.

APPENDICES

APPENDIX A

SURVEY FORM

The following is a survey gathering data for a Master's thesis on Software development and maintenance time estimates.
Please answer every question as accurately as possible and feel free to comment.

1. Questions 1-12 are general.

2. Questions 13-25 deal with the development of a system.

3. Questions 26-38 deal with modifications.

Use your most recent project for which you feel you can give the most accurate information. The system should have gone through at least one modification. Fill out a separate Part III for each unique modification.

NOTE: In Parts II and III, an answer of '1' indicates agreement or a very positive answer and an answer of '6' indicates disagreement and a negative answer.
Your time and energy are greatly appreciated.

Thank you,

Barbara Campbell
M.S. 555, K-1580
6-1935

PART I—General-Developmental

1. On the average, how many years experience in analysis did the software analyst(s) have? _____

2. On the average, how many years experience in design did the software designer(s) have? _____

3. On the average, how many years programming experience did the programmer(s) have? _____

4. How many people worked on the project?

   1. In analysis?_____

   2. In design?_____

   3. In programming?_____

5. What percentage of effort was spent in each of the following? (the total should equal 100)

   1. Analysis ___

   2. Design ___

   3. Initial Coding Effort ___

   4. Debugging and Testing ___

6. The average length - number of lines - of a module (procedure, paragraph or subroutine) was: ____

7. The approximate number of modules was: ____

8. What type system was it? (check those that are appropriate)

   1. Data Acquisition ___

   2. Report Generation ___

   3. Systems ___

   4. Online ___

   5. Batch ___

   6. Process Control ___

9. Were there timing constraints? _____

10. Were there memory constraints? _____

11. What computer language was used? _____

12. What kind of classification restrictions were associated with the system?

    1. none whatsoever ___

    2. the documentation or/and code were confidential ___

    3. the documentation or/and code were secret ___

    4. the output/input was confidential ___

    5. the output/input was secret ___

PART II--Specific-Developmental

	1	2	3	4	5	6	N/A
13. The software system was complex	___	___	___	___	___	___	___
14. The hardware interfaces were complex	___	___	___	___	___	___	___
15. There was a 'good' software library available •	___	___	___	___	___	___	___
16. Outside consultants were readily available	___	___	___	___	___	___	___
17. Manuals were available and understandable	___	___	___	___	___	___	___
18. The analysis was very structured	___	___	___	___	___	___	___
19. The design was very structured	___	___	___	___	___	___	___
20. The code was very structured	___	___	___	___	___	___	___
21. The objectives were well defined by the customer	___	___	___	___	___	___	___
22. The modules were cohesive	___	___	___	___	___	___	___
23. The system interacted heavily with existing software	___	___	___	___	___	___	___

24. The entire (all running tasks or load modules) software system was:

    1. < 32K ___
    2. 32K - 128K ___
    3. 128K - 512K ___
    4. 512K - 1Mk ___
    5. > 1Mk ___
    6. Other ___

25. What was the approximate time (in man-hours) for development and implementation? _____

PART III—Modifications

Fill out a Part III for each (at least one) unique modification.

	1	2	3	4	5	6	N/A
26. The person making the change was familiar with the software system	___	___	___	___	___	___	___
27. Complete and up-to-date documentation was available	___	___	___	___	___	___	___
28. Documentation was updated to show this change	___	___	___	___	___	___	___
29. The change required hardware change(s)	___	___	___	___	___	___	___
30. The software maintenance person was experienced in analysis	___	___	___	___	___	___	___
31. The software maintenance person was experienced in design	___	___	___	___	___	___	___
32. The software maintenance person was experienced in programming	___	___	___	___	___	___	___
33. The existing system was flexible	___	___	___	___	___	___	___
34. The development system was advantageous to making changes	___	___	___	___	___	___	___
35. The change was very complex	___	___	___	___	___	___	___

36. The change involved:

1. Adding module(s) ___
2. Adding lines ___
3. Changing lines ___
4. Change in logic ___
5. Deleting lines ___
6. Deleting module(s) ___
7. Other ___

37. The change was:

1. < 10 lines ___
2. 10 - 50 lines ___
3. 1 - 2 modules ___
4. 3 - 5 modules ___
5. > 5 modules ___
6. Other ___

38. What was the approximate time (in man-hours) to implement this change? ___

APPENDIX B

OBJECTIVES REPORT

Software Objectives Report

for

Software Manpower Estimate System

Barbara A. Campbell

Software Engineering

Introduction

The purpose of this system is to design a software system to store software project information and to produce man-hour estimates for development and modifications of software projects. By storing data, it is hoped that the algorithms used to estimate manhours may be updated regularly using this data.

A system survey has been omitted since current estimates are done using knowledge based on experience of past projects and estimates (seat-of-the-pants). Few software engineering groups have formal, proven methods of making software estimates.

This document consists of a list of objectives, their respective purposes and the proposed plan of implementation. The system will be implemented in two stages. The first stage will store historical data and analyze it. The second stage will develop an algorithm for estimating manhours for a project. The purpose of all data collection in phase I, is for analysis. A keyed database will be used to store the data for easy retrieval.

Objectives

The following represents the objectives, the reasoning behind each and a proposed method for implementation.

Phase I

1. Collect and store the following data per project:
    1. Type of system
    2. Timing constraint requirements
    3. Memory constraint requirements
    4. Computer language
    5. Classification
    6. Complexity
    7. Software library availability
    8. Consultant availability
    9. Manual availability
    10. Objectives definition
    11. Existing system interface
    12. Personnel experience
    13. Number of people
    14. Job organization
    15. Structure
2. Collect the following per modification for project:
    1. Familiarity with project
    2. Existing documentation

3. Updating of documentation

4. Affect on hardware

5. Personnel experience

6. System flexibility

7. Ease of use of development system

8. Complexity

9. Extent and content

3. Collect the following per project:

1. Project size

2. Change size

4. The following reports will be produced:

1. Project size report including: number of lines of code, number of manhours, number of people working on a project, sorted by number of lines of code.

2. Classification report: sorted by classification first and lines of code next and also reporting manhours.

3. Type of system

4. Timing constraints

5. Memory constraints

6. Complexity measures

7. Hardware interfaces

8. Library

9. Consultant

10. Manuals

11. Definition of objectives

12. Existing system interface

13. Personnel experience

14. Structure

15. Cohesion

16. Familiarity

17. Existing documentation

18. Flexibility

19. Extent

Reports of each variable and system length and system development time, sorted by the unique variable, will be generated. From these reports, analysis will be done to generate a qualitative relationship between the independent variable and the production time. The size of the system must also be reported for use as a controlling factor. Those variables found to be insignificant will be dropped from the system.

Data has been catagorized according to:

1. data which is known at the outset of a project, i.e., information that is known at the time at which a cost estimate is required;

2. data which surfaces during the development of a software system and may be used in the updating and refining of estimates but which would not be available in making the original estimate;

3. data known, relating to a modification, before a modification is done.

Module 1.1

Purpose: To store basic data

Get SYSTEM_BASICS from user
Store SYSTEM_BASICS in SYSTEM_BASICS_FILE

Module 1.2

Purpose: To store modification data

Get MODIFICATIONS from user
Store MODIFICATIONS in MODIFICATIONS_FILE

Module 2.1

Purpose: To generate and print a report of manhours and a requested data item sorted by the requested item.

```
For each PROJECT_NAME
        Get SYSTEM_BASICS from SYSTEM_BASICS_FILE
        Put SYSTEM_BASICS into SORT_FILE
Sort SORT_FILE on specified ATTRIBUTE
For each PROJECT_NAME
        Print PROJECT_NAME + ATTRIBUTE_VALUE + NUM_LINES_OF_CODE
                + NUM_HOURS_TO_PRODUCE
```

Module 2.2

Purpose: To generate and print a report of manhours and a requested data item sorted by the requested item.

```
For each PROJECT_NAME
        Get MODIFICATIONS from MODIFICATIONS_FILE
        Put MODIFICATIONS into SORT_FILE
Sort SORT_FILE on specified ATTRIBUTE
For each PROJECT_NAME
        Print PROJECT_NAME + ATTRIBUTE_VALUE + NUM_HOURS_TO_PRODUCE
```

Module 3.1

Purpose: To calculate the correspondence ratios for various attributes

```
Initialize all counters to zero
For each PROJECT_NAME
    Set NUMBER_OF_PROJECTS = NUMBER_OF_PROJECTS + 1
    Get SYSTEM_BASICS from SYSTEM_BASICS_FILE
    For each TYPE_OF_SYSTEM
        If TYPE_OF_SYSTEM of SYSTEM_BASICS = TYPE_OF_SYSTEM
                Increment NUMBER_TYPE (TYPE_OF_SYSTEM) by 1
                If MEMORY_CONSTRAINTS = YES
                        Increment NUMBER_MEMORY (TYPE_OF_SYSTEM) by 1
                If TIMING_CONSTRAINTS = YES
                        Increment NUMBER_TIMING (TYPE_OF_SYSTEM) by 1
                For each LANGUAGE
                        If LANGUAGE from SYSTEM_BASICS = LANGUAGE
                            Increment NUMBER_LANGUAGE (TYPE_OF_SYSTEM) by 1
                For each ATTRIBUTE
                    For each ATTRIBUTE_VALUE
                        If VALUE_OF_ATTRIBUTE = ATTRIBUTE_VALUE
                            Increment NUMBER_OF_ATTRIBUTE
                                (ATTRIBUTE, TYPE_OF_SYSTEM) by 1
For each TYPE_OF_SYSTEM
    Print TYPE_OF_SYSTEM
    For each ATTRIBUTE
        For each ATTRIBUTE_VALUE
                Set CORRESPONDENCE_RATIO = NUMBER_OF_ATTRIBUTE
                        (ATTRIBUTE, TYPE_OF_SYSTEM) / NUMBER_TYPE
                        (TYPE_OF_SYSTEM)
                Print CORRESPONDENCE_RATIO
                Put CORRESPONDENCE_RATIO into the ESTIMATION_FILE
                        using ATTRIBUTE
```

Module: 3.2

Purpose: To calculate and store the weighting factors for each attribute.

```
Initialize all counters to zero
For each PROJECT_NAME
    For I = 1 to NUMBER_OF_ATIRIBUTES
        Set NUMBER_OF_PROJECTS = NUMBER_OF_PROJECTS + 1
        Get SYSTEM_BASICS from SYSTEM_BASICS_FILE
        Set TOTAL_HOURS = TOTAL_HOURS + NUM_HOURS_TO_PRODUCE
        For J = 1, RANGE_OF_VALUES
            If VALUE_OF_ATIRIBUTE = J then
                Set ATIRIBUTE_NUM (I,J) = ATIRIBUTE_NUM (I,J) + 1
                Set ATIRIBUTE_NUM_HOURS (I,J) = ATIRIBUTE_NUM_HOURS (I,J)
                    + NUM_HOURS_TO_PRODUCE)
For I = 1 to NUMBER_OF_ATIRIBUTES
    For J = 1 to RANGE_OF_VALUES
        Set AVERAGE = ATIRIBUTE_NUM_HOURS (I,J)/ATIRIBUTE_NUM (I,J)
        Set AVERAGE_TOTAL = TOTAL_HOURS / NUMBER_OF_PROJECTS
        Set WEIGHT (I,J) = AVERAGE / AVERAGE_TOTAL
        Calculate the least squares fit curve using WEIGHT(I,J) and
            ATIRIBUTE_NUM(I,J) giving COEFFICIENTS
        Put COEFFICIENTS into ESTIMATION_FILE using I
```

Module 3.3

Purpose: To calculate a software cost estimate given various data about a project.

```
Initialize all counters to zero
Get PROJECT_NAME from user
Get SYSTEM_BASICS from SYSTEM_BASICS_FILE using PROJECT_NAME
Set CORE_ESTIMATE = NUM_LINES_OF_CODE
Set ESTIMATE = CORE_ESTIMATE
For each ATTRIBUTE
        Get COEFFICIENTS + CORRESPONDENCE_RATIO from ESTIMATION_FILE using
                ATTRIBUTE
        Calculate WEIGHT using COEFFICIENTS, VALUE_OF_ATTRIBUTE,
                CORRESPONDENCE_RATIO
        Set ESTIMATE = ESTIMATE + WEIGHT
Copy CORE_ESTIMATE + ESTIMATE to ESTIMATION_REPORT
Print ESTIMATION_REPORT
```

Module 4.0

Purpose: To compare experimental results with real data to analyze accuracy of model.

```
Initialize all counters to zero
For each PROJECT_NAME
        Get SYSTEM_BASICS from SYSTEM_BASICS_FILE
        Generate an estimate
        Set DIFFERENCE = NUM_HOURS_TO_PRODUCE - ESTIMATE
        Copy PROJECT_NAME + NUM_HOURS_TO_PRODUCE + ESTIMATE +
                DIFFERENCE to ANALYSIS_REPORT
        Set TOTAL_DIFFERENCE = TOTAL_DIFFERENCE + DIFFERENCE
        Set NUMBER_OF_PROJECTS = NUMBER_OF_PROJECTS + 1
        Set SUM_OF_SQUARES = SUM_OF_SQUARES + (DIFFERENCE ** 2)
Set STANDARD_DEVIATION = square root((SUM_OF_SQUARES -
    (TOTAL_DIFFERENCE ** 2)) / (NUMBER_OF_PROJECTS - 1)
Copy STANDARD_DEVIATION to ANALYSIS_REPORT
```

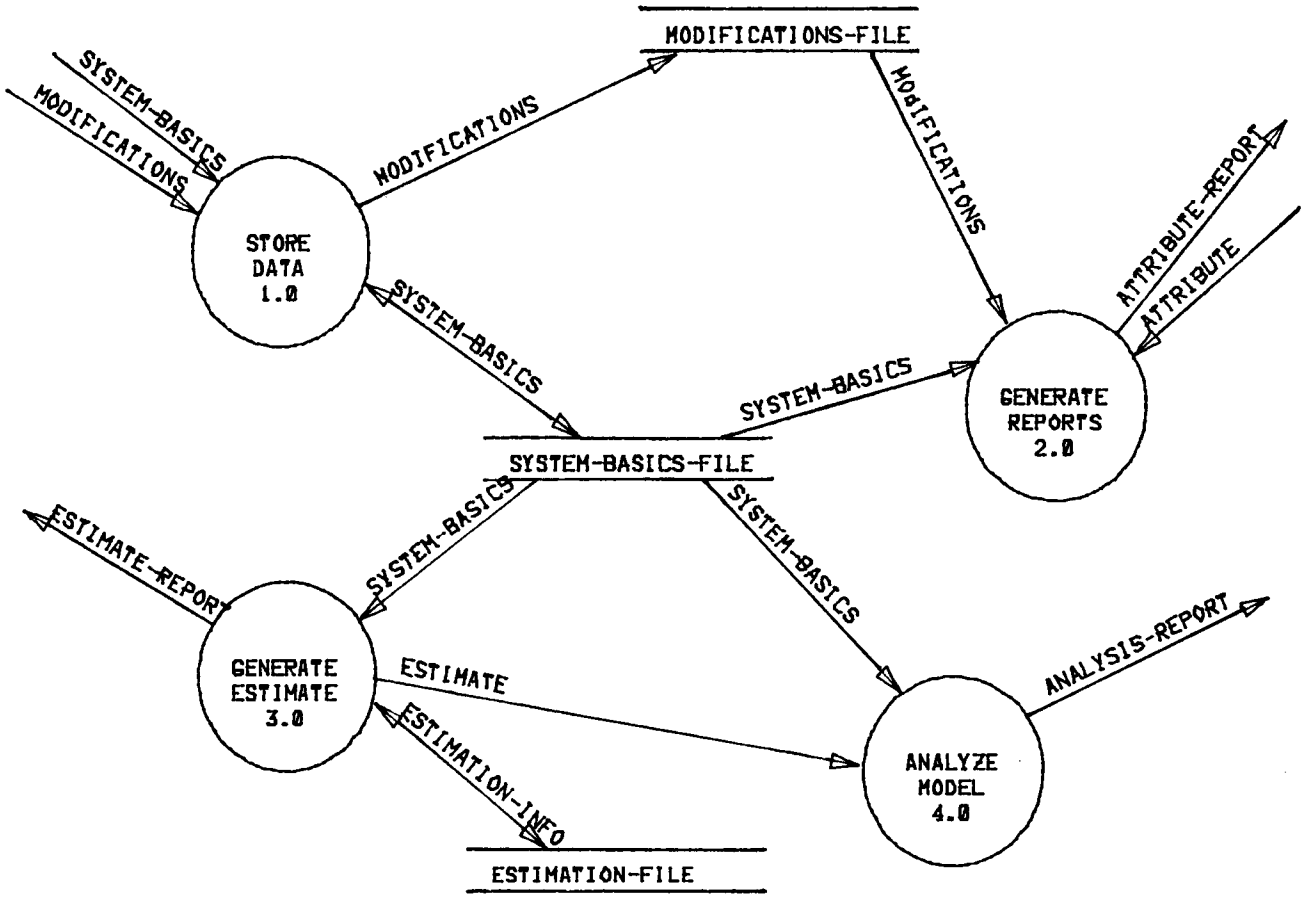

MODIFICATIONS-FILE
SYSTEM-BASICS
MODIFICATIONS
MODIFICATIONS
STORE DATA 1.0
MODIFICATIONS
MODIFICATIONS
ATTRIBUTE-REPORT
ATTRIBUTE
SYSTEM-BASICS
SYSTEM-BASICS
GENERATE REPORTS 2.0
SYSTEM-BASICS-FILE
ESTIMATE-REPORT
SYSTEM-BASICS
SYSTEM-BASICS
GENERATE ESTIMATE 3.0
ESTIMATE
ANALYSIS-REPORT
ANALYZE MODEL 4.0
ESTIMATION-INFO
ESTIMATION-FILE

DIAGRAM 0

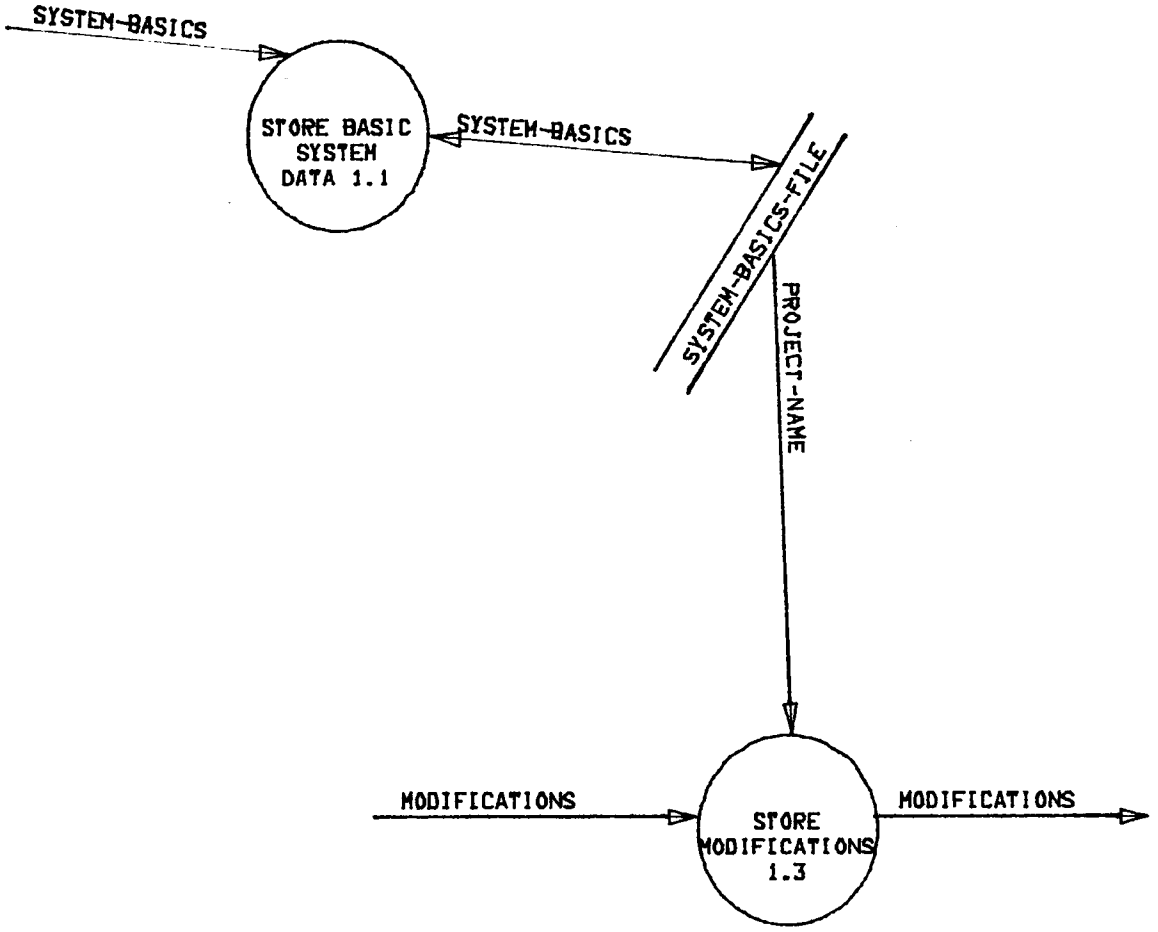

SYSTEM-BASICS
STORE BASIC SYSTEM DATA 1.1
SYSTEM-BASICS
SYSTEM-BASICS-FILE
PROJECT-NAME
MODIFICATIONS
STORE MODIFICATIONS 1.3
MODIFICATIONS

DIAGRAM 1

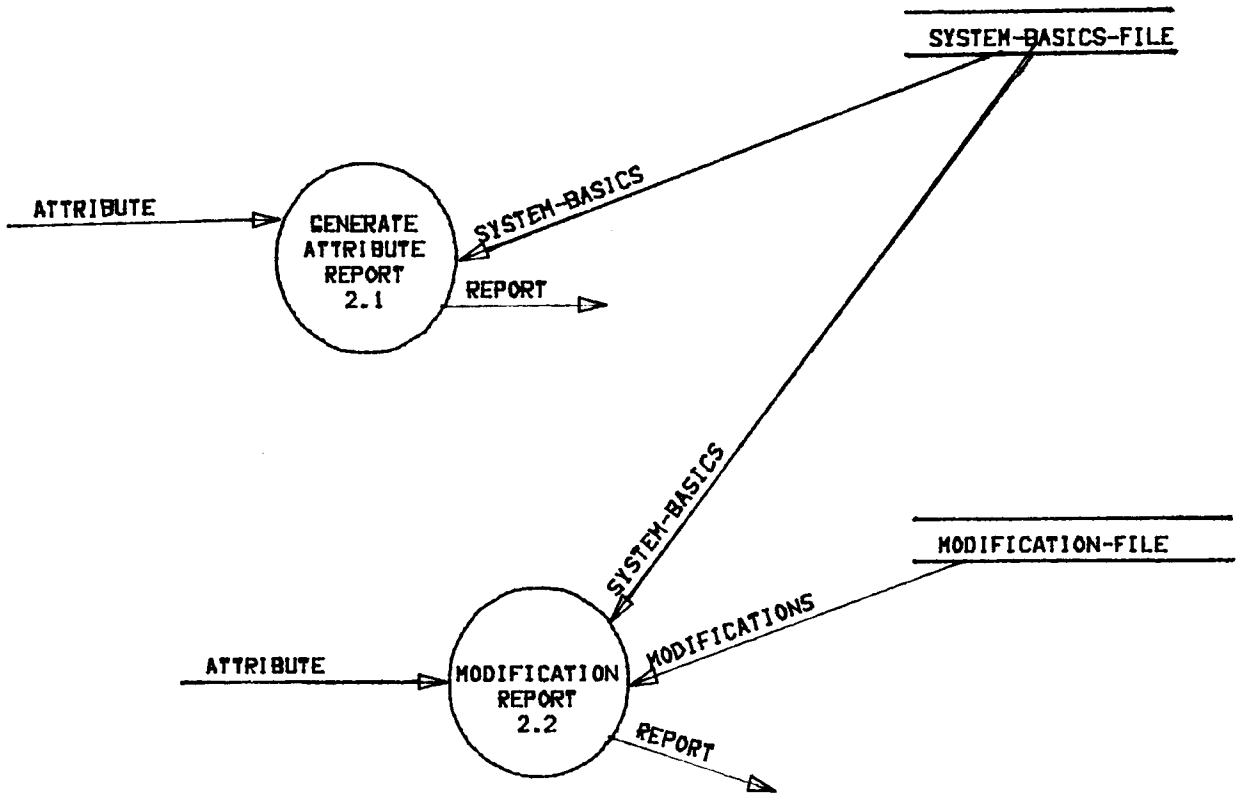

SYSTEM-BASICS-FILE
ATTRIBUTE
GENERATE ATTRIBUTE REPORT 2.1
SYSTEM-BASICS
REPORT
SYSTEM-BASICS
MODIFICATION-FILE
ATTRIBUTE
MODIFICATION REPORT 2.2
MODIFICATIONS
REPORT

DIAGRAM 2

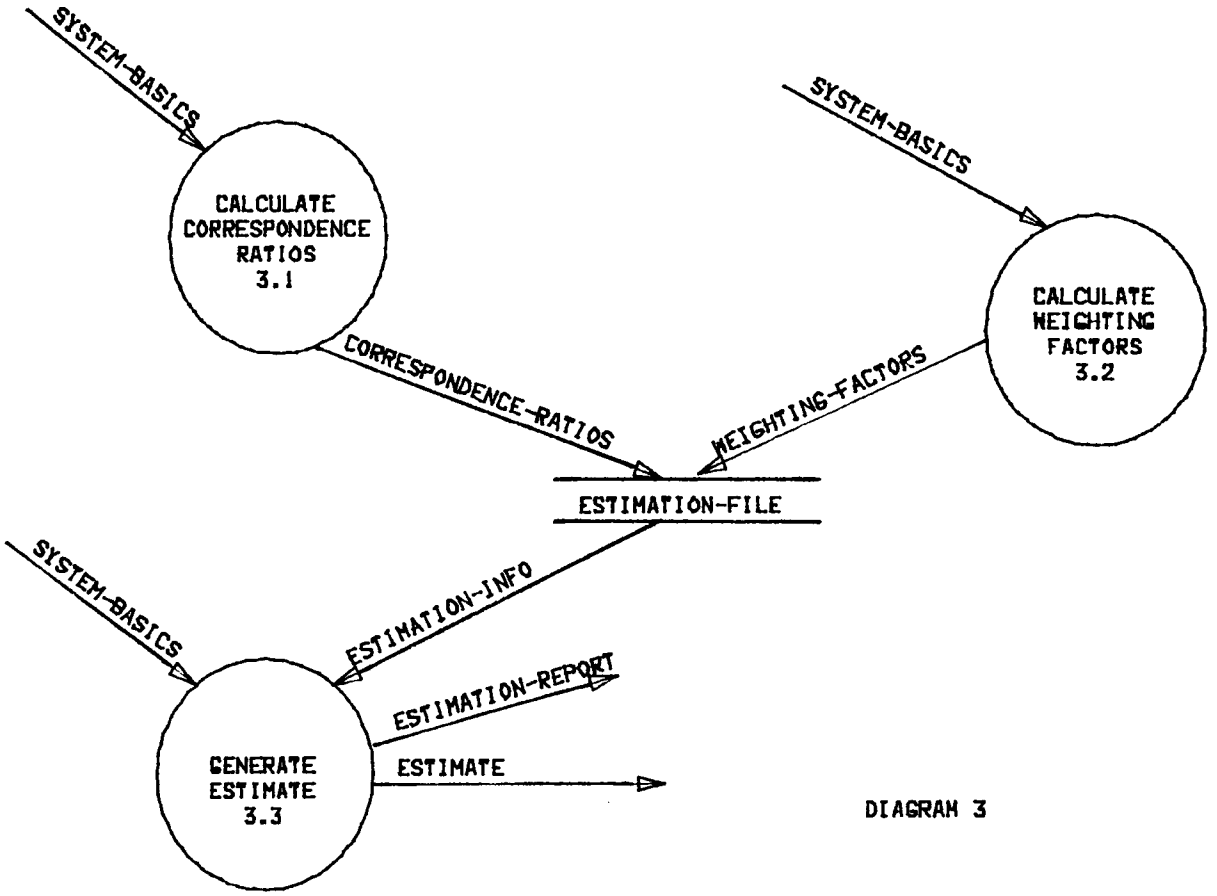

SYSTEM-BASICS
CALCULATE CORRESPONDENCE RATIOS 3.1
SYSTEM-BASICS
CALCULATE WEIGHTING FACTORS 3.2
CORRESPONDENCE-RATIOS
WEIGHTING-FACTORS
ESTIMATION-FILE
SYSTEM-BASICS
ESTIMATION-INFO
ESTIMATION-REPORT
ESTIMATE
GENERATE ESTIMATE 3.3

DIAGRAM 3

FILE LAYOUT

Location: Disk

File Name: COEFFS.DAT

File Size: 182 records

Record Size: 7 real (4 byte) words

File Description:

Record	Description
1	COEFFICIENTS for attribute 1
2	COEFFICIENTS for attribute 2
.	.
.	.
.	.
29	COEFFICIENTS for attribute 29
30	blank
31	CORRESPONDENCE_RATIO for attribute 1
32	CORRESPONDENCE_RATIO for attribute 2
.	.
.	.
.	.
59	CORRESPONDENCE_RATIO for attribute 29
60	blank
61	Errors for development COEFFICIENTS, attribute 1
62	Errors for development COEFFICIENTS, attribute 2
.	.
.	.
.	.
89	Errors for development COEFFICIENTS, attribute 29
90	blank
.	.
.	.
.	.
100	blank
101	Modification COEFFICIENTS, attribute 1
102	Modification COEFFICIENTS, attribute 2
.	.
.	.
.	.
141	Modification COEFFICIENTS, attribute 41
142	Errors for modification COEFFICIENTS, attribute 1
143	Errors for modification COEFFICIENTS, attribute 2
.	.
.	.
.	.
182	Errors for modification COEFFICIENTS, attribute 41

Record

Real Word	Description
1	COEFFICIENT for constant
2	COEFFICIENT for x
3	COEFFICIENT for x squared
4	COEFFICIENT for x cubed
5	blank
6	blank
7	blank

or

1	CORRESPONDENCE_RATIO for TYPE 1
2	CORRESPONDENCE_RATIO for TYPE 2
.	
.	
.	
7	CORRESPONDENCE_RATIO for TYPE 7

DATA
DICTIONARY

TERM	DEFINITION
ADL	*Add lines*
ADM	*Add modules*
AFFECT_ON_HARDWARE	*Measure of dependence of a software change on hardware*
ANALYSIS	*Measure of structure of analysis phase*
ANALYSIS_EXPERIENCE	*Average number of years experience of Analyst(s) on project*
ANALYSIS_PERCENTAGE	*Percentage of job done in analysis*
ANALYSIS_REPORT	{PROJECT_NAME + NUM_HOURS_TO_PRODUCE + ESTIMATE + DIFFERENCE} + STANDARD_DEVIATION
ATTRIBUTE	[CONCEPTS | HARDWARE_INTERFACES | LIBRARY | CONSULTANT | MANUALS | OBJECTIVES | ANALYSIS | DESIGN | CODE]
ATTRIBUTES	{ ATTRIBUTE }
ATTRIBUTE_NUM	*The number of times an attribute of a certain value occurs*
ATTRIBUTE_NUM_HOURS	*Given that a particular attribute has a particular value, the number of manhours on that project*
ATTRIBUTE_VALUE	[1 | 2 | ... | 6] *The value of an attribute*
AVERAGE	*The average length of time to produce systems with a specific attribute*
AVERAGE_MODULE_LENGTH	*Average length of a module, subroutine,
AVERAGE_TOTAL	*The average length of time to produce a system*
BAS	*Basic*
BTC	*Batch*

DATA
DICTIONARY

TERM	DEFINITION
CHANGE_SIZE	NUM_LINES_OF_CODE + NUM_HOURS_TO_PRODUCE
CHL	*Change lines*
CLASSIFICATION	[SEC | I/OS | CON | I/OC | UNC] *Security classification*
CLG	*Change in logic*
COB	*Cobol*
CODE_PERCENTAGE	*Percentage of job done in code*
COHESION_MEASURE	*Measure of the strength of functional association of elements within a module*
COMPLEXITY	SIZE + CONCEPTS + HARDWARE_INTERFACES * General difficulty of understanding of the system*
CON	*I/O and code is confidential*
CONCEPTS	*Ideas and data flow and manipulation of software*
CONSULTANT	*Measure of quantity and quality of consultants*
CORE_ESTIMATE	ESTIMATE
CORRESPONDENCE_RATIO	*A ratio of the number of times an attribute occurs within a system type with the number of times a type of system occurs*
DAS	*Data Acquisition System*
DEBUG_AND_TEST_PERCENTAGE	*The percentage of job done in debugging and unit testing and integration testing*
DESIGN	*Measure of structure of design phase*

DATA
DICTIONARY

TERM	DEFINITION
DESIGN_EXPERIENCE	*Average number of years experience of Designer(s) on project*
DESIGN_PERCENTAGE	*Percentage of job done in design*
DEVELOPMENT_SYSTEM	*Measure of ease of use of the system used to make modifications*
DIFFERENCE	*The difference between the calculated estimate and the actual amount of time taken*
DLL	*Delete lines*
DLM	*Delete modules*
ESTIMATE	*The estimated number of hours to do a job*
ESTIMATION_FILE	WEIGHTS + CORRESPONDENCE_RATIOS
ESTIMATION_REPORT	*A report of the core estimate with the adjusted estimate*
EXISTING_DOCUMENTATION	*Measure of quality of existing software documentation*
EXIST_SYS_INTERFACE	*The quantity of effort assigned the integrating the system with existing system(s)*
EXTENT	[ADM | ADL | CHL | CLG | DLL | DLM]
FAMILIARITY	*Measure of a persons familiarity with a system*
FLEXIBILITY	*Measure of flexibility of existing system with respect of modifications*
FOR	*Fortran*
GRA	*Graphics*
HARDWARE_INTERFACES	*Measure of connective I/O between user software and hardware*

DATA
DICTIONARY

TERM	DEFINITION
I	*An index*
I/OC	*I/O is confidential*
I/OS	*I/O is secret*
INFO_ESTIMATED	PROJECT_NAME + PROJECT_SIZE + CHANGE_SIZE
J	*An index*
JOB_ORGANIZATION	ANALYSIS_PERCENTAGE + DESIGN_PERCENTAGE + CODE_PERCENTAGE + DEBUG_AND_TEST_ PERCENTAGE
LANGUAGE	[COB | FOR | PAS | PL1 | BAS]
LIBRARY	*Measure of quantity and quality of software library available*
MANUALS	*Measure of qunatity and quality of available manuals*
MEMORY_CONSTRAINTS	[YES | NO]
MODIFICATION	PROJECT_NAME+FAMILIARITY+EXISTING_ DOCUMENTATION + UPDATING_DOCUMENTATION + AFFECT_ON_HARDWARE+ANALYSIS_EXPERIENCE + DESIGN_EXPERIENCE+PROGRAMMING_EXPERIENCE + FLEXIBILITY + DEVELOPMENT_SYSTEM + COMPLEXITY + EXTENT + CHANGE_SIZE
MODIFICATIONS	{ MODIFICATION }
MODIFICATIONS_FILE	MODIFICATIONS
MODULARITY	AVERAGE_MODULE_LENGTH + COHESION_MEASURE
NUMBER_LANGUAGE	*The number of projects within a type that are in a particular language*
NUMBER_MEMORY	*The number of projects within a type that have memory constraints*

DATA
DICTIONARY

TERM	DEFINITION
NUMBER_OF_ANALYSTS	*Number of analysts on project*
NUMBER_OF_ATTRIBUTE	*The number of times an attribute occurs within a type of system*
NUMBER_OF_ATTRIBUTES	*The number of attributes*
NUMBER_OF_DESIGNERS	*Number of designers on project*
NUMBER_OF_PROGRAMMERS	*Number of programmers on project*
NUMBER_OF_PROJECTS	*The number of projects in the data base*
NUMBER_TIMING	*The number of projects within a type that have timing constraints*
NUMBER_TYPE	*An array of integers for storing the number of systems of a particular type*
NUM_BYTES	*Size of system in bytes*
NUM_HOURS_TO_PRODUCE	*Number of man hours*
NUM_LINES_OF_CODE	*Number of lines of code*
OBJECTIVES	*Measure of how well the objectives were defined at the beginning of the project*
ONL	*Online, timesharing*
PAS	*Pascal*
PEOPLE_FACTORS	PERSONNEL_EXPERIENCE + NUMBER_OF_PEOPLE
PERSONNEL_EXPERIENCE	ANALYSIS_EXPERIENCE + DESIGN_EXPERIENCE + PROGRAMMING_EXPERIENCE
PL1	*PL1*
PRC	*Process Control*

DATA
DICTIONARY

TERM	DEFINITION
PROGRAMMING_EXPERIENCE	*Average number of years experience of programmer(s) on project*
PROJECT_NAME	*A 4 ASCII character name for a project; to be used as a key*
PROJECT_SIZE	NUM_LINES_OF_CODE + NUM_BYTES + NUM_HOURS_TO_PRODUCE
RANGE_OF_VALUES	6 *The range of values an attribute can have*
RPT	*Report generating system*
SEC	*I/O and code is secret*
SIZE	*Number of lines of code*
SORT_FILE	*A work file for sorting*
STANDARD_DEVIATION	*Standard deviation*
STRUCTURE	ANALYSIS + DESIGN + CODE
SUM_OF_SQUARES	*The sum of all the differences squared*
SYS	*System programming*
SYSTEM_BASICS	PROJECT_NAME + 2{TYPE_OF_SYSTEM}2 + TIMING_CONSTRAINTS + MEMORY_CONSTRAINTS + CLASSIFICATION + COMPLEXITY + LIBRARY + CONSULTANT + MANUALS + OBJECTIVES + EXIST_SYS_INTERFACE + PROJECT_SIZE + PEOPLE_FACTORS + JOB_ORGANIZATION + STRUCTURE + MODULARITY
SYSTEM_BASICS_FILE	{ SYSTEM_BASICS }
TIMING_CONSTRAINTS	[YES | NO]
TOTAL_DIFFERENCE	*The total of all the differences*
TOTAL_HOURS	*Total hours of all projects in the data base*

DATA
DICTIONARY

TERM	DEFINITION
TYPE_OF_SYSTEM	[DAS | RPT | GRA | SYS | ONL | BTC | PRC]
UNC	*Unclassified*
UPDATING_DOCUMENTATION	*Measure of quality of documentation to reflect changes and modifications*
VALUE_OF_ATTRIBUTE	[1 | 2 | ... | 6] *The value of an attribute*
WEIGHT	*A calculated weighting factor for an attribute*
WEIGHTS	NUMBER_OF_ATTRIBUTES { RANGE { WEIGHT } RANGE_OF_VALUES } NUMBER_OF_ATTRIBUTES
YES	*The affirmative*

Legend:
** means: comment [] means: either/or items separated by |
+ means: and () means: optional, may or may not be present
{} means: iterations of the data inside the braces

APPENDIX C

SOFTWARE

The code in this appendix is a sample of the major portion of code used. The modules are summarized below:

CORESP - calculates the correspondence ratios for the development attributes

DECODA - decodes the ASCII character string stored in the data base into its respective character and numberic components; as well as grouping components which are not valued in the 1 to 6 range

IDECDE - decodes a single digit into an integer

LSQFIT - calculates a weighted least squares fit using the polynomial method. Presently, it will only return coefficients up to a third order polynomial, this could easily be expanded.

WEIGT - calculates the weights for each attribute and calls LSQFIT to find the least squares fit third order polynomial

DEVEST - gives a development estimate for a specified project or a report of all projects, with the estimated time, actual time, error and percent error

Other modules used but not included in this appendix are:

MODIF - gives an estimate for a modification; it is similar to DEVEST, but uses selected attributes from both the development group and the modifications group

MWGHT - calculates the weights for all attributes - development and modification and fits the data to a curve, similar to WEIGT

DECODM - decodes the modification record from the data base; similar to DECODA

Also included is some sample code used in developing the data base:

1. the commands used to define the records, domains, and tables
2. a sample report generator

```
      PROGRAM CORESP
C
C  Purpose :  To calculate the correspondence ratios for various
C               attributes
C
      IMPLICIT INTEGER (A-Z)
      COMMON /BLK2/ ATTRIB,RECBUF,DUMMY
      INTEGER*4 SIZE,NUMLIN,NUMBYT,NUMHRS
      REAL ATTRAT(7,30)
C
      LOGICAL*1 KEYBUF(6), RECBUF(79),DUMMY(31)
      CHARACTER*6 PROJ
C
      DIMENSION ATTRIB (41), NUMATT(7,30)
      PARAMETER (     IRCSIZ = 79,
     +                KEYSIZ = 6,
     +                LUN    = 1,
     +                KEYNUM = 0,
     +                MODE   = 1)
      DATA KEYBUF/'0',5*0/
C
      NUMPRJ = 0
      DO 15 J = 1, 7
          DO 10 I = 1,30
                      NUMATT (J,I) = 0
                      ATTRAT (J,I) = 0.
10            CONTINUE
15    CONTINUE
C
      CALL RMSOPE (LUN,'[6,6]PROJREC.DAT',,,ISTAT)
      CALL RMSCON (LUN,KEYNUM,MODE,RECBUF,IRCSIZ,ISTAT)
      CALL RMSRAC (LUN,MODE,ISTAT)
C
      OPEN (UNIT=2,FILE='COEFFS.DAT',TYPE='OLD',DISP='SAVE',
     +          RECL=7,ACCESS='DIRECT',FORM='UNFORMATTED')
20    CALL RMSKEY (LUN,KEYNUM,KEYBUF,KEYSIZ,1,ISTAT)
      CALL RMSGET (LUN,IRFA,ISTAT)
C
      IF (ISTAT .EQ. -1472) GO TO 160
C
      CALL DECODA (PROJ,NUMLIN,NUMBYT, NUMHRS)
      NUMPRJ = NUMPRJ + 1
      DO 100 J = 1, 7
          IF ((ATTRIB(1) .EQ. J) .OR. (ATTRIB(2) .EQ. J)) THEN
              NUMATT (J,1) = NUMATT (J,1) + 1
```

```
                IF (ATTRIB(3) .EQ. 1) NUMATT(J,3) = NUMATT(J,3) + 1
                IF (ATTRIB(4) .EQ. 1) NUMATT(J,4) = NUMATT(J,4) + 1
                DO 90 I = 5,29
                    IF (ATTRIB(I) .GE. 4) NUMATT (J,I) = NUMATT(J,I) + 1
90              CONTINUE
            ENDIF
100     CONTINUE
        DO 150 I = 1, 6
                KEYBUF (I) = RECBUF (I)
150     CONTINUE
        GO TO 20
C
160     DO 300 J = 1, 7
            DO 200 I = 3,29
                IF (NUMATT(J,1) .NE. 0) THEN
                    ATTRAT (J,I) = FLOATI(NUMATT(J,I))/FLOATI(NUMATT(J,1))
                ENDIF
200         CONTINUE
300     CONTINUE
        CALL RMSCLO (LUN,ISTAT)
        DO 500 J = 31,60
                WRITE (2'J) (ATTRAT(I,J-30),I=1,7)
500     CONTINUE
        CLOSE (UNIT=2,DISP='SAVE')
3000    FORMAT (X,'ISTAT=',I4)
5000    FORMAT (X,7(E12.6,XX))
        CALL EXIT
        END



        SUBROUTINE DECODA (PROJ,NUMLIN,NUMBYT,NUMHRS)
C
C  Purpose :  To decode the record buffer into its various attributes
C
        IMPLICIT INTEGER (A-Z)
        COMMON /BLK2/ATTRIB,RECBUF,DUMMY
        INTEGER*4 SIZE,NUMLIN,NUMBYT,NUMHRS
        LOGICAL*1 RECBUF(79),DUMMY(31)
        CHARACTER*3 TYPE1,TYPE2,COMLAN,CLASS,TYPES(7),LANG(5),
    +           CLASIF(5)
        CHARACTER*6 PROJ
C
        DIMENSION ATTRIB(41)
        DATA TYPES/'SYS','PRC','DAS','ONL','GRA','BTC','RPT'/
        DATA LANG/'FOR','COB','PAS','PL1','BAS'/
        DATA CLASIF/'IOC','IOS','SEC','CON','UNC'/
C
C Put zeros in blank spaces, then decode
C
        DO 20 I = 7, 79
                IF (RECBUF(I) .EQ. ' ') RECBUF (I) = '0'
20      CONTINUE
```

```
      DECODE (6,1000,RECBUF(1))PROJ
      DECODE (3,1003,RECBUF(7))TYPE1
      DECODE (3,1003,RECBUF(10))TYPE2
      ATTRIB(3) = IDECDE(RECBUF(13),PROJ)
      ATTRIB(4) = IDECDE(RECBUF(14),PROJ)
      DECODE (3,1003,RECBUF(15))COMLAN
      DECODE (3,1003,RECBUF(18))CLASS
      DECODE (6,1006,RECBUF(21))SIZE
      IF (FLOATJ(SIZE) .GE. 32000.) THEN
            ATTRIB(7) = 30000
      ELSE
            ATTRIB(7) = SIZE
      ENDIF
      DO 21 I = 1,7
            ATTRIB (7+I) = IDECDE(RECBUF(26+I),PROJ)
21    CONTINUE
      DO 22 I = 1,10
            DECODE (2,1002,RECBUF(34+(I-1)*2))ATTRIB(14+I)
22    CONTINUE
      DO 23 I = 1,3
            ATTRIB (24+I) = IDECDE (RECBUF(53+I),PROJ)
23    CONTINUE
      DECODE (3,1013,RECBUF(57))ATTRIB(28)
      ATTRIB(29) = IDECDE (RECBUF(60),PROJ)
      DECODE (6,1006,RECBUF(61))NUMLIN
      DECODE (7,1007,RECBUF(67))NUMBYT
      DECODE (6,1006,RECBUF(74))NUMHRS
C
C  Attribute 1 and 2 are TYPEs of system
C
      DO 50 J = 1, 7
          IF (TYPE1 .EQ. TYPES(J)) THEN
              ATTRIB(1) = J
          ENDIF
          IF (TYPE2 .EQ. TYPES(J)) THEN
              ATTRIB(2) = J
          ENDIF
50    CONTINUE
C
C  Attribute 3 is time constraints
C
      IF (ATTRIB(3) .NE. 1) THEN
            ATTRIB(3) = 2
      ENDIF
C
C  Attribute 4 is memory constraints
C
      IF (ATTRIB(4) .NE. 1) THEN
            ATTRIB(4) = 2
      ENDIF
      DO 60 I = 1,5
C
C  Attribute 5 is the language used
```

```
C
            IF (LANG(I) .EQ. COMLAN) THEN
                ATTRIB (5) = I
            ENDIF
C
C  Attribute 6 is the security classification
C
            IF (CLASIF(I) .EQ. CLASS) THEN
                ATTRIB(6) = I
            ENDIF
60      CONTINUE
C
C  Attribute 7 is number of modules
C
        DO 130 J = 1,7
            IF ((ATTRIB(7).LT.(J*600)).AND.(ATTRIB(7).GE.((J-1)*600)))
     +          ATTRIB(7) = J
C
C  Attributes 15 - 20 are:
C       15 - average years experience in analysis
C       16 - average years experience in design
C       17 - average years experience in code
C       18 - number of people doing analysis
C       19 - number of people doing design
C       20 - number of people doing code
C
            DO 70 I = 15,20
                IF ((ATTRIB(I).LT.(J*3)).AND.(ATTRIB(I) .GE. ((J-1)*3)))
     +              ATTRIB (I) = J
70          CONTINUE
C
C  Attributes 21 - 24 are:
C       21 - percentage of time spent on analysis
C       22 - percentage of time spent on design
C       23 - percentage of time spent on code
C       24 - percentage of time spent on debugging
C
            DO 105 I = 21,24
                IF ((ATTRIB(I) .LT. (J*12)) .AND.
     +              (ATTRIB(I) .GE. ((J-1)*12))) THEN
                    ATTRIB (I) = J
                ENDIF
105         CONTINUE
C
C  Attribute 28 is the average length of a module
C
            IF ((ATTRIB(28) .LT. (J*25)) .AND.
     +          (ATTRIB(28) .GE. ((J-1)*25))) THEN
                    ATTRIB (28) = J
            ENDIF
130     CONTINUE
C
1000    FORMAT (A6)
```

```
1002    FORMAT (I2)
1003    FORMAT (A3)
1006    FORMAT (I6)
1007    FORMAT (I7)
1013    FORMAT (I3)
        RETURN
        END




        FUNCTION IDECDE (DIGIT,PROJ)
        CHARACTER*6 PROJ
        LOGICAL*1 DIGIT
        IF ((DIGIT .GE.,"60) .AND. (DIGIT .LE. ,"71)) THEN
                IDECDE = DIGIT - ,"60
        ELSE
                TYPE 100,PROJ,DIGIT
100             FORMAT (X,A6,X,O3)
                IDECDE = 0
        ENDIF
        RETURN
        END




        SUBROUTINE LSQFIT (NTERMS,NPOINT,COEFFS)
        COMMON /BLK1/X,ERROR,F,W,PJ
C
C  Purpose:  The purpose of this program is to perform a least squares
C            fit of a set of data points.
C
C  Arguments:    NTERMS  - input;  the degree of the desired polynomial
C                          approximation plus 1.
C             NPOINT  - input;  the number of data points to be used
C                          in the approximation
C             X       - input;  the x values
C             F       - input;  the experimental function value
C             WEIGHT  - input;  a positive weighting factor
C             ERROR   - output; the difference at each point between
C                          the experimental value F(x) and the calculated
C                          value P(x) using the approximating polynomial
C             PJ      - output;
C             COEFFS  - output; the coefficients
C
C
        DIMENSION X(20), F(20), W(20), ERROR(20),
     +          RJM1(20), PJ(20),
     +          B(10), C(10), D(10), S(10), COEFFS(5)
C
```

```
C  Initialize everything to zero
C
        DO 10 I = 1, NTERMS
                B(I) = 0.
                C(I) = 0.
                D(I) = 0.
                S(I) = 0.
10      CONTINUE
        C(1) = 0.
C
C  Calculate the constant by summing the weighted function values and
C  then dividing by the total weights.
C  NOTE:  If all weights are the same, then the number is just the
C       average of the functions.
C
        DO 20 I = 1, NPOINT
                D(1) = D(1) + F(I) * W(I)
                B(1) = B(1) + X(I) * W(I)
                S(1) = S(1) + W(I)
20      CONTINUE
        D(1) = D(1) / S(1)
C
C  Calculate the error for the polynomial of degree 0 ( a constant)
C
        DO 30 I = 1, NPOINT
                ERROR(I) = F(I) - D(1)
30      CONTINUE
        IF (NTERMS .EQ. 1) GO TO 100
C
C  if a polynomial is desired of degree greater than 1, continue,
C  and find the weighted average of the x's.
C
        B(1) = B(1) / S(1)
        DO 40 I = 1, NPOINT
                PJM1(I) = 1.
                PJ(I) = X(I) - B(1)
40      CONTINUE
        J = 1
C
C  Repeat
C
50      J = J + 1
        DO 60 I = 1, NPOINT
                P = PJ(I) * W(I)
                D(J) = D(J) + ERROR(I) * P
                P = P * PJ(I)
                B(J) = B(J) + X(I) * P
                S(J) = S(J) + P
60      CONTINUE
        D(J) = D(J) / S(J)
C
C  Find the error
C
```

```
        DO 70 I = 1, NPOINT
                ERROR (I) = ERROR (I) - D(J) * PJ(I)
70      CONTINUE
C
C  if J is equal to degree of the approximating polynomial
C  then return
C
        IF (J .EQ. NTERMS) GO TO 100
C
C  otherwise, continue
C
        B(J) = B(J) / S(J)
        C(J) = S(J) / S(J-1)
        DO 80 I = 1, NPOINT
                P = PJ(I)
                PJ(I) = ( X(I) - B(J) ) * PJ(I) - C(J) * PJM1(I)
                PJM1(I) = P
80      CONTINUE
C
C  Until J equals NTERMS
C
        GO TO 50
100     COEFFS(4) = D(4)
        COEFFS(3) = D(3) - D(4)*(B(1)+B(2)+B(3))
        COEFFS(2) = D(2) - D(3)*(B(1)+B(2))+D(4)*(C(1)-C(2)+B(1)*B(2)
     +                   +B(2)*B(3)+B(1)*B(3))
        COEFFS(1) = D(1) - B(1)*D(2) + D(3)*(B(1)*B(2)-C(1)) +
     +                   D(4)*(B(1)*B(2)*B(3)+B(3)*C(1)+B(1)*C(2))
        RETURN
        END




        PROGRAM WEIGT
C
C  Purpose :  To calculate the weights for various
C             attributes
C
        IMPLICIT INTEGER (A-Z)
        COMMON /BLK1/X,ERROR,F,W,PJ
        COMMON /BLK2/ATTRIB,RECBUF
        INTEGER*4 NUMLIN,NUMBYT,NUMHRS
        REAL WEIGHT (7,30), AVE,AVETOT, NRMHRS, X(20),
     +          ATTNHR(7,30), TOTHRS, ERROR(20), F(20), W(20),
     +          COEFFS(5), PJ(20)
C
        LOGICAL*1 KEYBUF(6), RECBUF(79), FILSPC(30)
        CHARACTER*6 PROJ
C
        DIMENSION ATTRIB(30), ATTNUM(7,30)
        PARAMETER (     IRCSIZ = 79,
```

```
     +                    KEYSIZ = 6,
     +                    LUN    = 1,
     +                    KEYNUM = 0,
     +                    MODE   = 1)
          DATA KEYBUF/'0',5*0/
C
C  Initialize everything to zero
C
          TOTHRS = 0
          NUMPRJ = 0
          DO 7 I = 1, 30
                  DO 6 J = 1, 7
                          ATINUM (J,I) = 0
                          ATINHR (J,I) = 0
6                 CONTINUE
7         CONTINUE
          CALL RMSOPE (LUN,'PROJREC.DAT',,,ISTAT)
          CALL RMSCON (LUN,KEYNUM,MODE,RECBUF,IRCSIZ,ISTAT)
          CALL RMSRAC (LUN,MODE,ISTAT)
          OPEN (UNIT=4,NAME='COEFFS.DAT',ACCESS='DIRECT',SHARED,RECL=7,
     +          FORM='UNFORMATTED',TYPE='OLD',DISP='SAVE')
10        CALL RMSKEY (LUN,KEYNUM,KEYBUF,KEYSIZ,1,ISTAT)
          CALL RMSGET (LUN,IRFA,ISTAT)
C
C If end of file, start processing information
C
          IF (ISTAT .EQ. -1472) GO TO 155
C
          CALL DECODA (PROJ,NUMLIN,NUMBYT,NUMHRS)
C
C  Find the normalized - number of hours per line of code - number of
C  hours to produce the code, this will account for the difference in
C  size of the projects
C
          NRMHRS = FLOATJ(NUMHRS) / FLOATJ (NUMLIN)
          TOTHRS = TOTHRS + NRMHRS
          NUMPRJ = NUMPRJ + 1
C
C  Attribute 1 and 2 are TYPEs of system
C
          DO 25 I = 1,2
              DO 20 J = 1,7
                  IF (ATTRIB(I) .EQ. J) THEN
                      ATINUM(J,I) = ATINUM (J,I) + 1
                      ATINHR(J,I) = ATINHR (J,I) + NRMHRS
                  ENDIF
20            CONTINUE
25        CONTINUE
C
C  Attribute 3 is time constraints
C  Attribute 4 is memory constraints
C
          DO 30 I = 3,4
```

```
            IF (ATTRIB(I) .EQ. 1) THEN
                ATINUM(1,I) = ATINUM (1,I) + 1
                ATINHR(1,I) = ATINHR (1,I) + NRMHRS
            ELSE
                ATINUM(2,I) = ATINUM (2,I) + 1
                ATINHR(2,I) = ATINHR (2,I) + NRMHRS
            ENDIF
30      CONTINUE
        DO 130 J = 1,7
            DO 35 I = 5,29
C
C  Attribute 5 is the language used
C  Attribute 6 is the security classification
C
                IF (ATTRIB(I) .EQ. J) THEN
                        ATINUM (J,I) = ATINUM (J,I) + 1
                        ATINHR (J,I) = ATINHR (J,I) + NRMHRS
                ENDIF
35          CONTINUE
130     CONTINUE
C
C  Attribute 7 is number of modules
C
C  Attributes 8 - 14 are:
C       8  - complexity of concepts
C       9  - use of hardware
C       10  - library availability
C       11 - consultant availability
C       12 - availability of manuals
C       13 - adequacy of definition of objectives
C       14 - extent of interfacing with existing software
C
C  Attributes 15 - 20 are:
C       15 - average years experience in analysis
C       16 - average years experience in design
C       17 - average years experience in code
C       18 - number of people doing analysis
C       19 - number of people doing design
C       20 - number of people doing code
C
C  Attributes 21 - 24 are:
C       21 - percentage of time spent on analysis
C       22 - percentage of time spent on design
C       23 - percentage of time spent on code
C       24 - percentage of time spent on debugging
C
C  Attributes 25 - 27 are:
C       25 - adequacy of structured analysis
C       26 - adequacy of structured design
C       27 - adequacy of structured code
C
C  Attribute 28 is the average length of a module
C  Attribute 29 is the measure of cohesion
```

```
C
C  Get next record
C
        DO 150 I = 1, 6
              KEYBUF (I) = RECBUF (I)
150     CONTINUE
        GO TO 10
155     DO 165 I = 1, 29
            K = 0
            DO 160 J = 1, 7
                IF (ATINUM (J,I) .NE. 0) THEN
                    K = K + 1
                    AVE = ATINHR(J,I) / FLOATI(ATINUM(J,I))
                    AVETOT = TOTHRS / FLOATI(NUMPRJ)
                    WEIGHT(J,I) = AVE/AVETOT
                    X(K) = FLOATI(J)
                    W(K) = FLOATI(ATINUM(J,I))
                    F(K) = WEIGHT(J,I)
                ENDIF
160         CONTINUE
            CALL LSQFIT (4,K,COEFFS)
            WRITE (4'I)COEFFS
            J = I + 60
            WRITE (4'J) (ERROR(L),L=1,K)
165     CONTINUE
        CALL RMSCLO (LUN,ISTAT)
        IF (ISTAT .LT. 0) WRITE (5,3000) ISTAT
3000    FORMAT (X,'ISTAT=',I4)
        CLOSE (UNIT=4,DISP='SAVE')
        CALL EXIT
        END



        PROGRAM DEVEST
C
C  Purpose :  To calculate the time estimate for software development
C                 systems with various attributes
C
        IMPLICIT INTEGER (A-Z)
        COMMON /BLK2/ATIRIB,RECBUF,DUMMY
        INTEGER*4 SIZE,NUMLIN,NUMBYT,NUMHRS,NUMLNS
        REAL WEIGHT, COEFFS(7), ATIRAT (7,30), RATIO,TEMP,TOT,SUMM,
     +  STDEV,ERRSQU,SUMERR,ERROR,TIME,NRMHRS
C       DOUBLE PRECISION ERRSQU,SUMERR
C
        LOGICAL*1 KEYBUF(6), RECBUF(79),DOALL,DESIRE(6),FILSPC(30),DUMMY(31)
        CHARACTER*6 PROJ
C
        DIMENSION ATIRIB(41)
        PARAMETER (     IRCSIZ = 79,
     +                  KEYSIZ = 6,
```

```
     +                    LUN    = 1,
     +                    KEYNUM = 0,
     +                    MODE   = 1)
          DATA KEYBUF/'0',5*0/
C
C  Initialize everything
C
          ERRSQU = 0.
          SUMERR = 0.
          NUMPRJ = 0
C
C  Determine and open the proper data file
C
          WRITE (5,1)
1         FORMAT ('$INPUT FILE SPEC. OF DATA:"DEV:[G,M]XXXXXX.EXT;"')
          READ (5,2) (FILSPC(I),I=1,29)
2         FORMAT (29A1)
          DO 3 I = 1,29
                  IF(FILSPC(I) .EQ. "40) FILSPC(I)=0
3         CONTINUE
          CALL RMSOPE (LUN,FILSPC,,,ISTAT)
          CALL RMSCON (LUN,KEYNUM,MODE,RECBUF,IRCSIZ,ISTAT)
          CALL RMSRAC (LUN,MODE,ISTAT)
C
C  prompt for the desired project, if no project is requested,
C  then, do a report of all
C
          WRITE (5,1000)
          READ (5,1010) DESIRE
          IF (DESIRE(1) .EQ. ' ') THEN
                  DOALL = .TRUE.
          ELSE
                  DOALL = .FALSE.
                  DO 5 I = 1,6
                          KEYBUF(I) = DESIRE(I)
5                 CONTINUE
          ENDIF
C
          CALL ASSIGN (3,'DEVEST.RPT')
C
          OPEN (UNIT=4,FILE='COEFFS.DAT',ACCESS='DIRECT',SHARED,RECL=7,
     +          FORM='UNFORMATTED',TYPE='OLD',DISP='SAVE')
C
C  Get correspondence ratios from the coefficients file
C
          IF (DOALL) THEN
              DO 8 I = 1,182
                  READ (4'I) COEFFS
                  WRITE (3,2020)I,(COEFFS(K),K=1,7)
8             CONTINUE
          ENDIF
          DO 9 I = 31,59
                  READ (4'I) (ATTRAT(J,I-30),J=1,7)
```

```
9       CONTINUE
C
C  Get a record from the data base
C
10      CALL RMSKEY (LUN,KEYNUM,KEYBUF,KEYSIZ,1,ISTAT)
        CALL RMSGET (LUN,IRFA,ISTAT)
C
C If end of file, start processing information
C
        IF (ISTAT .EQ. -1472) GO TO 155
C
C Put zeros in blank spaces, then decode
C
        CALL DECODA (PROJ,NUMLIN,NUMBYT,NUMHRS)
C
C  Find the normalized - number of hours per line of code - number of
C  hours to produce the code, this will account for the difference in
C  size of the projects
C
        TOT = 0.
        SUMM = 0.
        NUMPRJ = NUMPRJ + 1
C
C  For each attribute calculate the normalized hours
C
        K = ATTRIB (1)
        DO 50 I = 1,29
C               IF ((I .GT.7).AND.(I.LT.16)) GO TO 50
C               IF ((I.GT.22).AND.(I.LT.28)) GO TO 50
C               IF (I.EQ.21) GO TO 50
                READ (4'I)COEFFS
C
C               Find the correspondence ratio, determine if there
C               was a significant correspondence and set the ratio
C               accordingly
C
                TOT = TOT + 1.
                RATIO = ABS (ATTRAT(K,I) - .5)
                IF ((RATIO .GT. 0.25) .AND. (RATIO .LT. 0.5)) THEN
                        RATIO = 1. - RATIO
                ELSE
                        RATIO = 1.
                ENDIF
C
C               Calculate the weight using the least squares fit
C               coefficients and the correspondence ratios
C
                WEIGHT = RATIO * (COEFFS(1) + COEFFS(2)*ATTRIB(I) +
     +                  COEFFS(3)*ATTRIB(I)**2+COEFFS(4)*
     +                  ATTRIB(I)**3)
                SUMM = SUMM + WEIGHT
50      CONTINUE
C
```

```
C  The number of lines of code is equal to the number of modules times
C  the number of lines of code per module (attribute 28)
C
         NRMHRS = SUMM/TOT
         TIME = NRMHRS * NUMLIN
         ERROR = TIME - FLOATJ(NUMHRS)
         SUMERR = SUMERR + ERROR
         ERRSQU = ERRSQU + ERROR**2
         TEMP = ERROR/FLOATJ(NUMHRS)
         WRITE (3,2000) PROJ,FLOATJ(NUMHRS),TIME,ERROR,TEMP
C
C  Get next record
C
         DO 150 I = 1, 6
                KEYBUF (I) = RECBUF (I)
150      CONTINUE
         IF (DOALL) GO TO 10
155      STDEV = SQRT (((NUMPRJ*ERRSQU)-(SUMERR**2))/(NUMPRJ*(NUMPRJ-1)))
         WRITE (3,2010) STDEV
         CALL RMSCLO (LUN,ISTAT)
         IF (ISTAT .LT. 0) WRITE (5,3000) ISTAT
1000     FORMAT (X,'Enter the project for which an estimate is desired,'/
     +           x,'Or, enter RETURN for all projects:'$)
1010     FORMAT (6A)
2000     FORMAT (X,A6,X,5(E12.6,X))
2010     FORMAT (/X,'STANDARD DEVIATION = ',E12.6,X//)
2020     FORMAT (X,I3,XX,7(E12.6,XX))
3000     FORMAT (X,'ISTAT=',I4)
         CLOSE (UNIT=4,DISP='SAVE')
         CALL EXIT
         END
```

```
DEFINE DICTIONARY PROJECT.DIC;
DEFINE RECORD PROJECT_REC USING
01 PROJECT_REC.
        05 PROJECT_NAME                  PIC X(6).
        05 SYSTEM_BASICS.
                10 TYPE1_OF_SYSTEM       PIC XXX
                   VALID IF TYPE1_OF_SYSTEM IN TYPE_SYS_TABLE.
                10 TYPE2_OF_SYSTEM       PIC XXX
                   VALID IF TYPE2_OF_SYSTEM IN TYPE_SYS_TABLE.
                10 TIMING_CONSTRAINTS    PIC X.
                10 MEMORY_CONSTRAINTS    PIC X.
                10 COMPUTER_LANGUAGE     PIC XXX
                   VALID IF COMPUTER_LANGUAGE IN LANGUAGE_TABLE.
                10 CLASSIFICATION        PIC XXX
                   VALID IF CLASSIFICATION IN CLASSIFICATION_TABLE.
                10 COMPLEXITY.
                        15 SIZE          PIC 9(6).
                        15 CONCEPTS      PIC 9.
                        15 HARDWARE_INTERFACES  PIC 9.
                10 LIBRARY               PIC 9.
                10 CONSULTANT            PIC 9.
                10 MANUALS               PIC 9.
                10 OBJECTIVES            PIC 9.
                10 INTER_EXIST_SYSTEMS   PIC 9.
        05 ITEMS_OF_POST_INT.
                10 PEOPLE_FACTORS.
                        15 PERSONNEL_EXPERIENCE.
                                20 ANALYSIS_EXP PIC 99.
                                20 DESIGN_EXP   PIC 99.
                                20 PROGRAMMING_EXP      PIC 99.
                        15 NUMBER_OF_PEOPLE.
                                20 ANALYSIS_NUM PIC 99.
                                20 DESIGN_NUM   PIC 99.
                                20 PROGRAMMING_NUM      PIC 99.
                10 JOB_ORGANIZATION.
                        15 ANALYSIS_PER          PIC 99.
                        15 DESIGN_PER            PIC 99.
                        15 CODE_PER              PIC 99.
                        15 DEBUG_TEST_PER        PIC 99.
                10 STRUCTURE.
                        15 ANALYSIS_ST           PIC 9.
                        15 DESIGN_ST             PIC 9.
                        15 CODE_ST               PIC 9.
                10 MODULARITY.
                        15 AVERAGE_MOD_LENGTH    PIC 999.
                        15 COHESION_MEASURE      PIC 9.
        05 INFO_ESTIMATED.
                10 PROJECT_SIZE.
                        15 NUM_LINES_OF_CODE     PIC 9(6).
                        15 NUM_BYTES             PIC 9(7).
                        15 NUM_HOURS_TO_PRODUCE  PIC 9(6).
;
DEFINE RECORD MODIFICATION USING
```

```
01 MODIFICATIONS.
                10 PROJECT_NAME                  PIC X(6).
                10 FAMILIARITY                   PIC 9.
                10 EXISTING_DOCUMENTATION        PIC 9.
                10 UPDATING_DOCUMENTATION        PIC 9.
                10 AFFECT_ON_HARDWARE            PIC 9.
                10 ANALYSIS_EXPERIENCE           PIC 9.
                10 DESIGN_EXPERIENCE             PIC 9.
                10 PROGRAMMING_EXPERIENCE        PIC 9.
                10 FLEXIBILITY                   PIC 9.
                10 DEVELOPMENT_SYSTEM            PIC 9.
                10 COMPLEXITY                    PIC 9.
                10 EXTENT                        PIC X(3)
                   VALID IF EXTENT IN EXTENT_TABLE.
                10 CHANGE_SIZE.
                      15 NUM_LINES               PIC 9(6).
                      15 NUM_HOURS               PIC 9(6).
;
DEFINE TABLE TYPE_SYS_TABLE
         DAS : "DATA ACQUISITION",
         RPT : "REPORT GENERATION",
         GRA : GRAPHICS,
         SYS : SYSTEMS,
         ONL : ONLINE,
         BTC : BATCH,
         PRC : "PROCESS CONTROL",
         ELSE "INVALID SYSTEM"
END-TABLE
DEFINE TABLE LANGUAGE_TABLE
         COB : COBOL,
         FOR : FORTRAN,
         PAS : PASCAL,
         PL1 : PL1,
         BAS : BASIC,
ELSE "INVALID"
END-TABLE
DEFINE TABLE EXTENT_TABLE
         ADM : "ADD MODULE(S)",
         ADL : "ADD LINES",
         CHL : "CHANGE LINES",
         CLG : "CHANGE IN LOGIC",
         DLL : "DELETE LINES",
         DLM : "DELETE MODULE(S)",
ELSE "INVALID"
END-TABLE
DEFINE TABLE CLASSIFICATION_TABLE
         SEC : "SECRET",
         IOS : "I/O SECRET",
         CON : "CONFIDENTIAL",
         IOC : "I/O CONFIDENTIAL",
         UNC : "UNCLASSIFIED",
ELSE "INVALID"
END-TABLE
```

```
DEFINE DOMAIN PROJREC USING PROJECT_REC ON PROJREC.DAT;
DEFINE DOMAIN MODIFI USING MODIFICATION ON MODIFI.DAT;
DEFINE FILE FOR PROJREC KEY=PROJECT_NAME (NO DUP);
DEFINE FILE FOR MODIFI KEY=PROJECT_NAME;



READY PROJREC
REPORT PROJREC SORTED BY TYPE1_OF_SYSTEM, TYPE2_OF_SYSTEM,
  NUM_LINES_OF_CODE ON TYPE.RPT
SET REPORT-NAME = "PROJECT TYPE REPORT"
SET LINES-PAGE = 60
SET COLUMNS-PAGE = 132
PRINT PROJECT_NAME, TYPE1_OF_SYSTEM, TYPE2_OF_SYSTEM,
  NUM_LINES_OF_CODE, NUM_HOURS_TO_PRODUCE
AT BOTTOM OF TYPE1_OF_SYSTEM PRINT SKIP, "AVERAGE HOURS=",
  AVERAGE (NUM_HOURS_TO_PRODUCE), SKIP
REPORT END
```

APPENDIX D

GLOSSARY

1. Computational complexity [11]--"quantitative aspects of the solutions to computational problems."

2. Difficulty gradient--rate of change of difficulty.

3. Entropy--the measure of disorderedness; when related to software, new systems that interact heavily with existing systems have the greatest entropy; complete rebuilds of existing independent systems have the least entropy; a new stand-alone system falls somewhere in between [30].

4. Psychological complexity--characteristics of software which make it difficult to understand and work with.

5. Software complexity metrics--the difficulty programmers experience in understanding and modifying software.

VITA

Barbara Ann Campbell was born in East Liverpool, Ohio, on April 3, 1956. She graduated from William Allen High School in 1974. Her freshman year of college was spent at Grove City College, she then transferred to and graduated from Bucknell University with a Bachelor of Science degree in Mathematics in 1978. In 1979, she took a job with Union Carbide Corporation, Nuclear Devision at the Oak Ridge Gaseous Diffusion Plant and began work in graduate school. She received her Master of Science degree with a major in Computer Science from The University of Tennessee, Knoxville, in December 1983.