Abstract

The ability to estimate the personnel time and costs required for the completion of programming and systems projects is an important managerial tool for the information systems department. This article presents a survey of the estimation techniques found in the literature by describing each technique and discussing its strengths and weaknesses. Some empirical evidence on how the various program and programmer/analyst characteristics affect project time and cost are also reported.

Keywords: Systems development, time and cost estimation
ACM Categories: 2.4, 3.5
With this complexity in mind, it is apparent that a meaningful assessment of any part of the systems development process can be made only when the variables that are changing at any one time are few in number. For this reason, an organization should not attempt to use the cost factors of others to estimate costs for its own development process. It should instead seek to measure productivity based on its own organizational context, its own hardware, the chosen source language, and programming mode, etc.

The next section presents the techniques that may be used for estimating programming variables, while those used for systems estimation follow.

**Estimation of Programming Time/Cost**

The methods that have been proposed in the literature for the estimation of programming time are presented in order of increasing sophistication. Hence, the early methods are easier to apply, but since they do not rely on an empirical base the estimates produced by them are subject to greater error. The later methods require a comprehensive databank on past projects, and substantive analysis in order to determine the variables that are relevant to the particular organization.

The methods which are presented are classified as follows:
1. Personal Experience
2. Analogy
3. Work Factors
4. Standards
5. Parametric Equations

**Personal experience**

This estimation method is the most subjective one. It is based on the judgment of the project supervisor, senior programming staff, or the programmers who will write the programs. Typically, supervisors take a weighted average of the various estimates, giving more weight to the judgment of experienced performers, or they use the different estimates to determine an upper and lower bound.

Greater reliability can be obtained by breaking the total effort into relatively small, manageable job units which are clearly delineated. The individual tasks are estimated separately, then aggregated to obtain the total project resources. An advantage of this method is the participation of the people who are going to perform the task, thus increasing the incentive to complete the task within the agreed time constraints.

A well-known technological forecasting approach, called the Delphi technique [19], could be used to improve the forecasting ability of individual estimators. Using this approach, a panel of estimators who remain anonymous, present independent estimates to a coordinator. The coordinator then provides each panel member with the distribution of group responses, such as the median, upper, and lower quartiles. Each panel member is asked to revise or support his estimates if his response differs from the group consensus. By iteration of the estimation/revision processes, the technique aims to reach a group convergence on the resource estimates.

**Analogy**

This method of estimation requires that fairly high level data on past development projects be retained by the organization. To employ this method at least one project with similar features must have been completed previously. The new project must be clearly specified at least at the functional level, permitting comparison of similar elements. Using the assumption that similar tasks require similar resources, an estimate is made of the matching areas, perhaps scaled upward to allow for unforeseen problems. Functions that cannot be compared are estimated separately by some other method.

The major problem encountered in the use of this method is that it cannot be applied to systems larger than that used for comparison. Aron [1] states that "system complexity grows as the square of the number of system elements; therefore, experience with a small system cannot account for all the things that will have to be done in a large system." In addition, this procedure cannot be applied to systems which are totally new in content.

**Work factors**

This method is a step toward the analytical method of estimation. It is based on an estimation equation, plus a set of factor tables which provide the variable values to be used as inputs to the equation [3, 11, 21, 24].

Weiss [24] has proposed the following estimation equation to find the total man-days required for a programming project (define program logic, code, compile, test/debug, and document):

\[
\text{Total Man-days} = \left( \frac{\text{Program complexity}}{\text{Input function complexity}} + \frac{\text{Output function complexity}}{\text{Programmer experience + Job knowledge}} \right) \times \left( \frac{\text{Program complexity}}{\text{Input complexity}} + \frac{\text{Output complexity}}{\text{Program complexity}} \right)
\]

In order to use the equation, Weiss provides a number of tables to determine the complexity factors associated with the particular programming assignment. For example, for input and output complexity, sequential tape files are given a factor

---

1Aron's [1] term for this procedure is the "Experience Method."

2Note that certain project management systems such as SPECTRUM-1 also provide estimation of programming time as part of their systems estimation process. SPECTRUM will be discussed more fully in the section on the total system development time and costs.
of 1, whereas a database file is assigned a factor of 4. For programmer experience the factors range from 0.5 for an experienced programmer to 4 for a programming trainee. Program complexity ratings are differentiated by program function at three levels of complexity. For example, an average edit function is assigned a rating of 2, a display function a rating of 1, and an average update function, a rating of 3.

Henney [10] evaluates the approach suggested by Weiss by stating:

*the advantages of this method, in principle, are that the estimate is built up on identifiable definitions of complexity for the program as a whole (although still relying on some subjective judgment), and there is no need to assess a global complexity factor, or to try and estimate the program length before it is written. The disadvantages are that it is complex, there is no evidence of its validation, and it purports to be a general purpose formula, which gives it little chance of particular success.*

More basic equations to calculate total programming time have also been developed. Watson [23] has proposed the following equation:

\[ \text{Total Man-days} = \frac{1}{4}SC + 2S + 4C + 2N \]

where C is a complexity rating ranging from 1 (easy) to 5 (highly complex), N is the number of record types, and S is the program size in units of 250 statements. The above result can then be adjusted for programming experience.

This approach has a number of problems. It requires an estimate of the number of lines of code, which in itself is a difficult estimating task. It does not give any guidelines as to the range of programming tasks, from very small to very complex, for which the formula is applicable. There is also no evidence as to the empirical basis for its development.

**Standards**

Two ways in which standards may be used to estimate programming costs will be presented here. A third method, described by Wolverton [25], employs standard costs for the total development life cycle. However, the procedure could be applied to the programming function alone.

**Lines of Code Per Hour**

Johnson [13] has suggested that the average number of lines of COBOL code (LOC) produced per hour may be used as a means of determining program development time. The average number of LOC/hour for small projects requiring less than 250 man-days was found to be 8.7, while projects requiring more than 500 man-days for development were found to have been produced at an average rate of 2.0 LOC/hour.

According to the following statement by Brooks [2], adjustment of these figures would suggest a 1 LOC/hour production rate for compilers and .3 LOC/hour for operating systems. Brooks states:

*My guideline in the morass of estimating complexity is that compilers are three times as bad as normal batch application programs, and operating systems are three times as bad as compilers.*

These figures, in fact, correspond well with those given by Brooks which are shown in Table 1.

| Table 1. Productivity Rates (LOC) |
|-------------------------------|----------------|----------------|
| Type of Development           | Johnson Hourly | Johnson Yearly |
| Batch                         | 3              | 6336          |
| Compiler                      | 1              | 2112          |
| Operating system              | 0.3            | 704           |
|                               |                |                |
|                               | Brooks Yearly  |                |
|                               | 6,000 - 9,000  |                |
|                               | 2,000 - 3,000  |                |
|                               | 600 - 800      |                |

34  *MIS Quarterly/June 1980*
Again, the difficulty in implementing this method lies in the need to calculate the number of expected LOC before programming takes place. However, if the system under consideration is to replace an existing computerized system, the estimate may be reasonably accurate.

**Module Standards**

This standard method of estimating programming costs is more analytical than the LOC method presented above. It requires an extensive review of past projects in order to obtain the variability in the number of statements per module and the programming rate for modules of a particular class. The statistics found by Donelson [7] for COBOL programs run on an IBM 360/40 batch processing computer are shown in Table 2.

By determining the number of modules of each class \( M_i \), the cost of programming can be estimated using the equation:

\[
\text{Programming Cost} = \sum_{i=1}^{12} M_i \left( \frac{S_i + \alpha_i \sigma S_i}{P_i + \beta_i \sigma P_i} \right) R_p
\]

where, \( M_i \) = number of modules of class \( i \),
\( S_i \) = mean number of statements per module per class \( i \),
\( \sigma S_i \) = standard deviation of \( S_i \),
\( \alpha_i \) = chosen multiple of \( \sigma S_i \) (planner’s estimate),

\( P_i \) = programming rate for modules of class \( i \).

**Table 2. Programming and Testing Estimates for COBOL Batch Applications**

<table>
<thead>
<tr>
<th>Module Class</th>
<th>Statements per module†</th>
<th>Programming Rate (statements per hour)</th>
<th>Computer test hours per module</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Standard Deviation</td>
<td>Mean</td>
</tr>
<tr>
<td>1. Data definition</td>
<td>62</td>
<td>52</td>
<td>16</td>
</tr>
<tr>
<td>2. One-time utility</td>
<td>177</td>
<td>92</td>
<td>30</td>
</tr>
<tr>
<td>3. Conversion utility</td>
<td>449</td>
<td>179</td>
<td>24</td>
</tr>
<tr>
<td>4. General utility</td>
<td>260</td>
<td>120</td>
<td>5</td>
</tr>
<tr>
<td>5. Database interface</td>
<td>450</td>
<td>150</td>
<td>10</td>
</tr>
<tr>
<td>“bridge”</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Edits</td>
<td>1715</td>
<td>415</td>
<td>16</td>
</tr>
<tr>
<td>7. Updates</td>
<td>1278</td>
<td>528</td>
<td>20</td>
</tr>
<tr>
<td>8. Processing</td>
<td>1186</td>
<td>108</td>
<td>8</td>
</tr>
<tr>
<td>9. Major extracts</td>
<td>530</td>
<td>29</td>
<td>15</td>
</tr>
<tr>
<td>10. Minor extracts</td>
<td>186</td>
<td>76</td>
<td>7</td>
</tr>
<tr>
<td>11. Major reports</td>
<td>907</td>
<td>436</td>
<td>8</td>
</tr>
<tr>
<td>12. Minor reports</td>
<td>260</td>
<td>95</td>
<td>9</td>
</tr>
</tbody>
</table>

*N/A = Not Available
†Does not include COPIED data definition statements
\[ P_{\hat{r}} = \text{programming statements per hour for module class } r, \]
\[ \sigma P_{\hat{r}} = \text{standard deviation of } P_{\hat{r}}, \]
\[ \beta_{\hat{r}} = \text{chosen multiple of } \sigma P_{\hat{r}}, \text{ and} \]
\[ R_p = \text{hourly charge for programming.} \]

The method is flexible, allowing for assessment of risk in the form of the multipliers, \( \kappa \) and \( \beta_r \), i.e., the estimator's uncertainty about the current development will be reflected in the number of standard deviations from the mean specified in the equation.

**Parametric equations**

This method involves the use of equations derived by the application of multiple regression techniques to estimate project development times. An example of the estimating equation may be:

\[
\text{Man-months} = \beta_0 + \beta_1 \times (\% \text{ Mathematical Instructions}) + \beta_2 \times (\text{Programming Language}) + \beta_3 \times (\text{Number of Subprograms}) + \beta_4 \times (\text{Experience Level of Programmer})
\]

In statistical notation, the general form of this equation is:

\[
y = \beta_0 x_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon
\]

where \( y \) is the value to be estimated, \( x_0 \) is a constant, \( x \)'s are the variables in absolute or coded form, \( \beta \)'s are the coefficients to be estimated, and \( \varepsilon \) is the error term. Examples of the use of the regression method to estimate programming project development times can be found in [5, 6, 9, 12, 15, 22].

The parametric equation as an estimation method is the one which looks at the problem most objectively. It is also the most difficult to develop. It requires a history file to be used in estimating the coefficients, and detailed statistical work to determine the parameters to be included in the equation. Once the coefficients are determined, the estimate for a new programming task can be calculated from the equation by substituting the variable values relating to the task.

There are a number of caveats which should be kept in mind when using multiple regression techniques. Nelson [15] describes these as follows:

Such work not only produces equations that can be used for estimating purposes, it also serves to identify factors that have statistically significant impact on expenditures, and hence directs management attention to those critical factors, many of which are subject to management control.

A disadvantage also arises, however, when the results of cost research are published in the form of equations. Such equations are frequently interpreted by the user as demonstrating a specific causative relationship. This is not the case. Equations developed by multiple regression techniques do reveal important parameters and may represent the relationships that provide the most statistically significant manner of describing the character of the analyst's sample. However, they do not necessarily represent natural law, as do many of the equations used by the engineer or the physicist. Also, they must be used in their entirety or not at all. If a value for any one of the independent variables were not available, the equation could not be used as it stands, since repeating the multiple regression analysis without the missing parameter would result in the reassignment of weights to all of the remaining variables and the \( Y \)-intercept of the equation.

Therefore, the transfer of the regression estimation equations published in the literature from the company whose data was used to another, is difficult. Variables such as company size, system design methodology, hardware, operating systems, etc., which might not have been taken into consideration in estimating the regression parameters would be likely to cause differences. Only in those cases where the company and system characteristics are similar can such equations be transferred.
Another consideration in using the regression equation for estimating purposes is the nature of the variables employed. Some equations contain variables such as number of classes of items in the database per lines of code, number of pages of delivered documentation per lines of delivered code, or total number of personnel divided by maximum number working at one time, which can only be determined ex-post, that is, after the completion of the programming task. However, equations which contain variables which are measured ex-post would still be useful in evaluating the comparative productivity of the programming task after project completion.

Estimating equations which contain quantitative rather than qualitative variables are more desirable. Quantitative variables such as the number of input/output files or number of reports generated can be measured objectively. Qualitative variables, such as complexity of the program or programmer experience with similar applications, must be coded using subjective judgment. Therefore, the use of quantitative variables only, would provide more theoretically sound equations.

There are not many published reports of parametric equations developed for estimation. Some of the variables which were included in those equations are shown in Table 3 to give the reader an indication of those factors which might have a possible impact on performance. An organization trying to develop an estimating equation could collect historical data on these variables, and consider them in its estimation process.

Although lines of code in a program are highly correlated with programming time (for example) see [12]), this variable was not included in Table 3 since it is difficult to estimate in advance. On a more macro level the number of application subprograms [18], and the number of COBOL paragraphs [12] could be used instead of lines of code.

Programmer experience seems a priori to be an important variable affecting programming time, and has been found to be so in various studies [5, 6, 9, 22]. Some authors who have not found experience to be significant [12, 25], suggest that less experienced programmers may be given the easier tasks, or that more experienced programmers may be promoted to systems analysts as possible reasons for such findings [12].

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Referenced By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Input Record Types</td>
<td>Gayle [9], Jeffery and Lawrence [12]</td>
</tr>
<tr>
<td>Number of Output Record Types</td>
<td>Gayle [9], Jeffery and Lawrence [12]</td>
</tr>
<tr>
<td>Number of Input Fields</td>
<td>Chrysler [5, 6]</td>
</tr>
<tr>
<td>Number of Input Edits</td>
<td>Chrysler [5, 6]</td>
</tr>
<tr>
<td>Number of Input Files</td>
<td>Chrysler [5, 6]</td>
</tr>
<tr>
<td>Number of Report Types</td>
<td>Putnam [18]</td>
</tr>
<tr>
<td>Number of Files</td>
<td>Chen [4], Jeffery and Lawrence [12], Putnam [18]</td>
</tr>
<tr>
<td>Number of Application Subprograms</td>
<td>Putnam [18]</td>
</tr>
<tr>
<td>Number of COBOL Paragraphs</td>
<td>Jeffery and Lawrence [12]</td>
</tr>
<tr>
<td>Programmer Experience</td>
<td>Chrysler [5, 6], Gayle [9], Walston and Felix [22]</td>
</tr>
</tbody>
</table>
Estimation of Total Systems Development Time/Cost

There are four principal ways in which costs or times for the whole of the systems development project may be estimated. These are:

1. Proportion of Life Cycle Method
2. Work Factors Method
3. Standards Method
4. Manpower Life Cycle Method

The first three of these methods may be regarded as micro-estimating approaches since they use program elements as basic building blocks, and hence bear a direct relationship to the methods presented for estimation of programming time. The fourth, the manpower life cycle method, is a totally different concept in that it takes a macro view of the total project. To date, reports on this approach in the literature have been confined to very large software development projects.

Before proceeding with a detailed discussion of the micro-estimating techniques note should be made that no technique makes specific mention of any overlap of the phases or iterations which may be required as the development proceeds. For example, a re-evaluation of systems design may be required after the commencement of the programming phase. Hence, the implementors, themselves should include such estimates in their calculations.

**Proportion of life cycle method**

Once the time to develop the programming phase of a project has been estimated, extrapolation provides an estimate for the total project. There are many reports in the literature showing fairly constant time periods associated with the different development phases. However, the individual organization would be well advised to establish its own phase times, as life cycle phases do not necessarily correspond from one organization to another. In addition, implementors may wish to break these phases into a greater number of subphases. The same basic method can be used with specified proportions of the total effort being assigned to the designated activities. Selected figures are given in Table 4.

An example of the use of this method is given by Donelson [7], who also calculates the cost of program keypunch time and computer test time. (Refer to page 35 and Table 2 for Donelson's basic parameters.) Total project cost is calculated by:

\[
\sum_{i=1}^{12} \frac{M \epsilon}{P \epsilon + \beta \epsilon \sigma P \epsilon} \cdot \left(1.1R_S + R_P\right) + \frac{S \epsilon + \alpha \epsilon \sigma S \epsilon}{125} \cdot R_K + T \epsilon \cdot R_C
\]

**Table 4. Percentage of Time Spent in Life Cycle Phases**

<table>
<thead>
<tr>
<th>Business Systems</th>
<th>Systems Analysis &amp; Design</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Donelson [7]</td>
<td>52%</td>
<td>48%</td>
</tr>
<tr>
<td>Johnson [13]</td>
<td>50%</td>
<td>Programming &amp; Testing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software Systems</th>
<th>Design</th>
<th>Implementation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aron [1]</td>
<td>30%</td>
<td>40%</td>
<td>30%</td>
</tr>
<tr>
<td>Wolverton [25]</td>
<td>Analysis &amp; Design</td>
<td>Coding &amp; Development</td>
<td>Checkout &amp; Test</td>
</tr>
<tr>
<td></td>
<td>40%</td>
<td>20%</td>
<td>40%</td>
</tr>
</tbody>
</table>
where, $R_S = \text{hourly charge for systems analysis and design};$

$R_K = \text{hourly charge for keypunching};$

$R_C = \text{hourly charge for computer test time};$

$T_i = \text{mean number of computer test hours for module of class } i$ (refer to the rightmost column in Table 2).

The expression on the left hand side of the plus sign is similar to the programming cost formula shown on page 35, except for the addition of the systems analysis and design time multiplier (1.1 $R_S$). The number 1.1 is the proportion of systems analysis and design time with respect to programming time (.52/.48), using the figures derived by Donelson as shown in Table 4. The figure 125 used on the right hand side of the expression stands for the average number of COBOL statements keypunched per hour. Note that the method is somewhat inconsistent in that it does not incorporate estimates of standard deviation for computer test time. This type of procedure may be used in association with any of those described for estimation of programming times.

**Work factors**

Work factors methods similar to those described earlier for the estimation of programming time, may be used also in the assessment of systems time. For example, Weiss [24] has suggested the following approach. The system development process is represented in seventeen steps, such as the design of system output, design of forms, approval of conversion program tests, and follow-up review. Each of these steps is rated as simple, simple/complex, complex, or very complex. A table provides the suggested man-days required for each step at each complexity level. For example, five man-days are allocated to developing very complex system test procedures while one man-day is allocated to designing master files for a simple system. The man-days assigned to a step are then multiplied by a weighting factor for analyst experience, ranging from 0.5 for an expert to 4.0 for one with limited systems experience, to obtain a final estimate. The sum of the estimates of all the steps represents the total system time.

The advantages and disadvantages of such an approach are similar to those discussed for programming estimation (see p. 34).

The project management system, SPECTRUM-1 [20], uses essentially the same approach as that discussed above. It provides time estimates for tasks in the form of average, minimum, and maximum figures. The task times are aggregated into figures for each of twelve phases. A number of adjustment factors may then be applied, including project size, average team experience, percentage of time on other work, user understanding, newness of techniques, etc. For example, the adjustment for a team composed of all senior people would be -10% of the calculated time, while a team of juniors would be permitted an increase of 40%; similar figures apply to well-defined user understanding as opposed to no user understanding.

**Standards**

Wolverton [25] has reported a software cost estimation technique based on the assumption that total development costs vary proportionally with the number of instructions. Using a “reverse” proportion of life cycle method, he then calculates costs for the individual phases of development.

The cost for the complete software development cycle is estimated at the program or subprogram level by consideration of the number of object instructions, the functions of the routine, and its relative difficulty. The six functions identified by Wolverton are control, input/output, pre/post processor, algorithm, data management, and time critical processes. Difficulty is defined as three levels, (easy, medium, and hard) for each of two categories, (old and new routines) producing a total of six levels. This approach envisages the use of standard costs per instruction for each of the above function-difficulty combinations (36 in all). Using these standard costs, plus the knowledge of program functions, difficulty level and estimated number of instructions per program, the total cost of software development is calculated. The next step is to break this total into its subphases in the following way: 8%; requirements; preliminary design 18%; interface definition 4%; detailed design 16%; code and debug 20%; development testing 21%; validation, testing, and operational demonstrations 13%.
There are three problems with this method. First, the building of a reliable databank is necessary since only data based on the organization’s own productivity can be employed. Second, any procedure which uses a set of standards makes the assumption that the building blocks are directly comparable, which is not necessarily true when the elements under consideration are computer programs. Third, estimation of the number of object instruction may be subject to considerable error, especially if the program is classified as “new,” or if the estimation is made before the detailed design phase.

Advantages claimed for this procedure include the minimal amount of data required to obtain a cost estimate, once the databank is established, and requirements are directly traceable to costs. In addition, the effect of alternative designs may be readily assessed.

**Manpower life cycle**

This method is based on a formal representation of manpower utilization rates over the project life cycle. The basic premise upon which the method rests is that certain facts about a software development project are known at the beginning of development. From these facts a graph of expected manpower utilization over time may be generated. It is a macro approach applicable only to large software projects from the time of detailed development.

Norden [16, 17] found that software development projects are composed of cycles corresponding to each phase of the life cycle, e.g., planning design, model (prototype), release (extension), and modification and maintenance. Each of these cycles is well described by the Rayleigh equation, given by:

\[ y = 2K \cdot \exp(-at^2) \]

where \( y \) = manpower utilized in each time period \( t \),

\( K \) = total cumulative manpower used on the project in units of man years,

\( a \) = shape parameter, governing the rate of change of manpower utilization, and

\( t \) = time in years.

In addition, the curves representing the cycles, when combined, form an overall development curve which also may be described by a Rayleigh equation. A graph of the equation is shown in Figure 2.

![Figure 2. Manpower Utilization Curve](image-url)
The Rayleigh equation was then applied to software development by Putnam [18]. He found that the time of maximum manpower utilization almost corresponds to the time when the system is first operational. Therefore, Putnam defined the time to peak as the development time of the project (t_d).

Extensive testing of the Rayleigh equation against a substantial number of past projects from several organizations showed that many had correlation coefficients in excess of 0.95, indicating that the equation can reproduce the practical manpower utilization situation extremely well [18].

In order to use the Rayleigh equation to predict total cost and manpower life cycle requirements, parameters K and t_d must be estimated. An example of the estimating approach as described by Putnam is discussed below. Putnam collected historical data on K (manpower utilized), t_d (development time), x_2 (number of reports), x_3 (number of application subprograms) from nineteen projects, and using multiple regression techniques, developed the following five estimation equations (constant terms in the equation were dropped):

\[
\begin{align*}
\frac{K}{t_d^2} &= 0.1991 x_2 + 0.0859 x_3 \\
\frac{K}{t_d} &= 0.2700 x_2 + 0.5901 x_3 \\
K &= -0.3447 x_2 + 2.7931 x_3 \\
Kt_d &= 4.1385 x_2 + 11.909 x_3 \\
Kt_d^4 &= -483.3196 x_2 + 791.08 x_3
\end{align*}
\]

In order to test the estimating power of these equations he chose one project from the nineteen used to generate the equation. This project had 179 reports (x_2) and 256 application subprograms (x_3). Substituting these values into the equations shown above, and simultaneously solving them for K and t_d, provided an estimate of K = 682.35 man years, and t_d = 3.4558 years (the actual values for the project were K = 700 and t_d = 3.65).

Once t_d is estimated, the shape parameter, a, can be obtained from the relationship:

\[
a = \frac{1}{2(t_d^2)}.
\]

Substituting for a, the Rayleigh curve is given by:

\[
y(t) = 2K \frac{t}{2t_d^2} \exp\left(-\frac{t^2}{2t_d^2}\right)
\]

and, for K = 682.35 and t_d = 3.4558, we obtain

\[
y(t) = 57.1 t \exp\left(-\frac{t^2}{23.89}\right).
\]

The above equation permits calculation of the number of man years needed for year t as shown in Table 5.

In summary, the method described above generates a number of estimating equations based on historical data and regression methods. To determine the manpower required for a new project in the same organization, first the number of reports and application subprograms are estimated. These values are then applied to the estimating equations, and values for K and t_d are generated. Using K and t_d in the Rayleigh curve, an estimate of manpower needed for each year of the project is obtained. Other more sophisticated approaches to estimation using the Rayleigh curves can be found in Putnam [18].

### Concluding Comments

The information systems department has the goal of providing management with the information it needs to conduct the organization's day-to-day operations, and the information it needs to direct the organization toward its short range and long range objectives. Although few information

---

<table>
<thead>
<tr>
<th>t (year)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>man years needed</td>
<td>52.1</td>
<td>96.6</td>
<td>117.5</td>
<td>116.9</td>
<td>100.3</td>
<td>75.9</td>
<td>51.4</td>
<td>...</td>
</tr>
</tbody>
</table>

---

*Table 5. Estimated Manpower Derived From the Rayleigh Equation*
system departments have reached that stage yet, they should at least by trying to work toward those objectives. Therefore, it follows that a department whose main objective is to provide information, must build its own information system to gather information on performances of systems analysts, programmers, computer operators, as well as on equipment and other needs. It seems that many information systems departments tend to ignore their own management control systems while trying to help the organization.

In order to achieve a good internal management system, subjective standards should be used less, in favor of more objective procedures. This may be achieved by starting to build a history file, retrieving the information needed, and analyzing it in order to develop standards or parametric equations for improved estimation methods.

The building of the history file is a prerequisite for developing any one of the estimation techniques described in this paper, except for the "personal experience" approach. A framework which outlines the data elements which, \textit{a priori}, are thought to be significant, is useful in order to determine what types of project data to collect. The framework suggested by Chrysler [5, 6] (see Figure 1), and a measurement database described by Walston and Felix [22] could be used as guidelines.

In analyzing the various methods described in this paper it is important to understand their differing characteristics. Methods such as the work factors and the manpower utilization curves are intended to be general purpose ones which could be used in any organizational context. However, one should be careful to realize that the manpower utilization curves were developed for very large software projects, thus, their validity can be relevant only in such environments. The work factors methods, since they try to be general in their orientation, would have less success in a particular organization than a parametric equation developed in the given organizational environment. Standards and parametric equation methods are presented in order to describe approaches an organization could use for developing estimation techniques rather than to provide the readers with specific numbers which they could use in their organizations.

A review of the literature for the last ten years shows that very little in terms of new methods has been proposed in this area. In our opinion, the methods available today are more than adequate for a company to establish an estimation approach. All that is needed is management's willingness to employ the planning and control philosophy used in other functional areas in the information systems department.

References


[10] Henney, A. "Techniques for Estimating Programming Effort and Assessing Pro-


**About the Authors**

Izak Benbasat is Assistant Professor of Accounting and MIS at the University of British Columbia. He received his M.S. and Ph.D. in MIS at the University of Minnesota. He is presently spending his sabbatical year at the MIS Research Center at the University of Minnesota.

Iris Vessey is currently Lecturer in Information Systems at the Department of Commerce, University of Queensland, Australia. She holds an M.Sc., Dip. of Info. Proc., and an MBA degree from the University of Queensland. Her current research interests include software design, software maintenance and the cognitive processes involved in computer programming and debugging.