Modeling the Behavior of Large Software Projects

Dana S. Borger
Mladen A. Vouk

Center for Communications and Signal Processing
Department of Computer Science
North Carolina State University

TR-91/19
November 1991
This report is based on the M.S. thesis written by Dana S. Borger under the direction of Mladen A. Vouk (NCSU, Dept. of Computer Science, May 1991).

The report describes an approach to modeling large software project behavior. The model assumes a phased software development process which is driven by milestone dates. The process is applied to overlapping projects that correspond to the releases of a single software product. Enhancements are implemented in the product as a large number of software components developed in parallel. A certain number of the components must complete each process phase for the product release to attain that phase milestone. The model incorporates Parkinson’s Law and the Deadline Effect. The product component life-cycle phase durations are treated as stochastic variables. The durations are related to the time available to a planned milestone date using a regression function. A linear regression model is used to demonstrate the model’s adequacy in an observed development environment. The model is evaluated using both empirical project data and simulations. Simulations also serve as a means of investigating non-analytical effects, such as schedule slippages. Empirical results indicate that the model can be used by software managers and software developers to predict the finish behavior of a project for any life-cycle phase. Such predictions could aid in analyzing risks associated with not meeting planned project deadlines. The model is expandable and programmable, helping to ensure that the model paradigms will be investigated further, and can be adapted and instituted in different environments.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF SYMBOLS AND ABBREVIATIONS</td>
<td>vi</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Goals of this Study</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Model Motivation</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Review of Past Work</td>
<td>9</td>
</tr>
<tr>
<td>1.3.1 Software Schedule Estimation Models</td>
<td>9</td>
</tr>
<tr>
<td>1.3.2 Parkinson’s Law and the Deadline Effect</td>
<td>11</td>
</tr>
<tr>
<td>1.4 Scope of this Study</td>
<td>14</td>
</tr>
<tr>
<td>1.5 Overview of the Study</td>
<td>15</td>
</tr>
<tr>
<td>2 MODEL DEVELOPMENT</td>
<td>17</td>
</tr>
<tr>
<td>2.1 Model Definitions and Assumptions</td>
<td>17</td>
</tr>
<tr>
<td>2.2 Detailed Description of the Development Environment</td>
<td>22</td>
</tr>
<tr>
<td>2.3 The Single-Variable Model</td>
<td>27</td>
</tr>
<tr>
<td>2.3.1 Model Notation and Definitions</td>
<td>27</td>
</tr>
<tr>
<td>2.3.2 Calculating the Distribution of Finish Times</td>
<td></td>
</tr>
<tr>
<td>2.3.3 The Expected Values of Durations and Finish Times</td>
<td></td>
</tr>
<tr>
<td>2.3.4 Applying Least Squares Linear Regression</td>
<td></td>
</tr>
<tr>
<td>2.3.5 Estimating the Variances of Duration and Finish Times</td>
<td></td>
</tr>
<tr>
<td>2.4 The Effect of Schedule Changes</td>
<td></td>
</tr>
<tr>
<td>3 MODEL EVALUATION</td>
<td>48</td>
</tr>
<tr>
<td>3.1 Model Performance on Simulated Data</td>
<td></td>
</tr>
<tr>
<td>3.1.1 A Simulation with No Schedule Changes</td>
<td>44</td>
</tr>
<tr>
<td>3.1.2 Simulating Effects of a Schedule Change</td>
<td></td>
</tr>
<tr>
<td>3.2 Using the Model to Describe Historical Data</td>
<td>48</td>
</tr>
<tr>
<td>3.2.1 Discussion of the Historical Data</td>
<td>48</td>
</tr>
<tr>
<td>3.2.2 Linear Regression Analysis</td>
<td>50</td>
</tr>
<tr>
<td>3.2.3 Another Data Partitioning Scheme</td>
<td>58</td>
</tr>
<tr>
<td>4 CONCLUSION</td>
<td>66</td>
</tr>
<tr>
<td>4.1 Summary</td>
<td>66</td>
</tr>
<tr>
<td>4.2 Recommendations for Future Research</td>
<td>68</td>
</tr>
<tr>
<td>5 REFERENCES</td>
<td>71</td>
</tr>
<tr>
<td>APPENDIX I: CONFIDENCE INTERVALS FOR LSQ FIT</td>
<td>73</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 3-1  Summary of input parameters for the simulation.............................. 42
Table 3-2  Summary of input parameters for the schedule slip simulation.............. 44
Table 3-3  Summary statistics for program finish time for programs in Project 1 ...... 50
Table 3-4  Regression summary for program duration vs. start time for Project 1 ...... 54
Table 3-5  Regression summary for program duration vs. start time for programs in Project 1, with the new partitioning of carry-over programs...................... 60
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1-1</td>
<td>Scatterplot of component duration vs. start time for components in Project 1</td>
<td>6</td>
</tr>
<tr>
<td>Figure 1-2</td>
<td>Histogram of start time (a) and finish time (b) for components in Project 1</td>
<td>7</td>
</tr>
<tr>
<td>Figure 1-3</td>
<td>Cumulative frequency plot for component finish times for components in Project 1</td>
<td>8</td>
</tr>
<tr>
<td>Figure 1-4</td>
<td>Example of a partition of a software developer’s time.</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2-1</td>
<td>The software development process.</td>
<td>23</td>
</tr>
<tr>
<td>Figure 2-2</td>
<td>Illustration of overlapping projects in the environment.</td>
<td>26</td>
</tr>
<tr>
<td>Figure 2-3</td>
<td>Defined linear relationship between program starting time and duration.</td>
<td>31</td>
</tr>
<tr>
<td>Figure 2-4</td>
<td>Illustration of re-applying the model after a schedule change.</td>
<td>38</td>
</tr>
<tr>
<td>Figure 3-1</td>
<td>Histogram of actual program start times in Project 1</td>
<td>41</td>
</tr>
<tr>
<td>Figure 3-2</td>
<td>Histogram of simulated program finish times.</td>
<td>42</td>
</tr>
<tr>
<td>Figure 3-3</td>
<td>Cumulative frequency plot of simulated program finish times.</td>
<td>43</td>
</tr>
<tr>
<td>Figure 3-4</td>
<td>Histogram of actual program finish times in Project 1</td>
<td>43</td>
</tr>
<tr>
<td>Figure 3-5</td>
<td>Cumulative frequency plot of actual program finish times in Project 1</td>
<td>44</td>
</tr>
<tr>
<td>Figure 3-6</td>
<td>Histogram of simulated program finish times, with no slip in the schedule.</td>
<td>46</td>
</tr>
<tr>
<td>Figure 3-7</td>
<td>Histogram of simulated program finish times, with a 5 unit slip in the schedule announced at time 0.</td>
<td>46</td>
</tr>
<tr>
<td>Figure 3-8</td>
<td>Histogram of all program duration for Project 1</td>
<td>49</td>
</tr>
<tr>
<td>Figure 3-9.1</td>
<td>Scatterplot of program duration vs. start time for Project 1</td>
<td>51</td>
</tr>
<tr>
<td>Figure 3-9.2</td>
<td>Results of fitting equation 3.1.</td>
<td>54</td>
</tr>
<tr>
<td>Figure 3-10</td>
<td>Histogram of the residuals from the full model regression for Project 1</td>
<td>56</td>
</tr>
<tr>
<td>Figure 3-11</td>
<td>Normal probability plot of the residuals from the full model regression for Project 1</td>
<td>57</td>
</tr>
</tbody>
</table>
LIST OF FIGURES  
(continued)

| Figure 3-12 | Scatterplot of the residuals vs. the predicted values from the full model regression for Project 1 | 57 |
| Figure 3-13 | Scatterplot of the Cook's Di vs. observation number from the full model regression for Project 1 | 58 |
| Figure 3-14 | Results of fitting equation 3.11 | 60 |
| Figure 3-15 | Histogram of residuals from the model in 3.11 for Project 1 with the new division of carry-over programs | 61 |
| Figure 3-16 | Normal probability plot of the residuals from the model in 3.11 for Project 1 with the new division of carry-over programs | 62 |
| Figure 3-17 | Scatterplot of the residuals vs. predicted values from the model in 3.11 for Project 1 with the new division of carry-over programs | 62 |
| Figure 3-18 | Plot of the residuals by group from the model in 3.11 for Project 1 with the new division of carry-over programs | 63 |
| Figure 3-19 | Plot of the Cook's Di statistic by group from the model in 3.11 for Project 1 with the new division of carry-over programs | 63 |
| Figure 4-1 | Example plots of times programs achieved various milestones versus the time they achieved the first milestone | 69 |
### LIST OF SYMBOLS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>SYMBOL / ABBREVIATION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>COCOMO</td>
<td>Constructive Cost Model.</td>
</tr>
<tr>
<td>SLIM</td>
<td>Putnam’s macro software sizing and estimating model.</td>
</tr>
<tr>
<td>X</td>
<td>Random component start time.</td>
</tr>
<tr>
<td>x or x&lt;sub&gt;i&lt;/sub&gt;</td>
<td>An observed component start time.</td>
</tr>
<tr>
<td>Y</td>
<td>Random component phase duration.</td>
</tr>
<tr>
<td>y or y&lt;sub&gt;i&lt;/sub&gt;</td>
<td>An observed component duration.</td>
</tr>
<tr>
<td>Z</td>
<td>Random component finish time.</td>
</tr>
<tr>
<td>z or z&lt;sub&gt;i&lt;/sub&gt;</td>
<td>An observed component finish time.</td>
</tr>
<tr>
<td>pmf</td>
<td>“probability mass function”</td>
</tr>
<tr>
<td>∞</td>
<td>Infinity.</td>
</tr>
<tr>
<td>N</td>
<td>Number of components in a software product.</td>
</tr>
<tr>
<td>U&lt;sub&gt;v&lt;/sub&gt; or U</td>
<td>V=v</td>
</tr>
<tr>
<td>P(U=u)</td>
<td>The probability that a random variable U takes the value u.</td>
</tr>
<tr>
<td>p&lt;sub&gt;U&lt;/sub&gt;(u)</td>
<td>The pmf of a random variable U evaluated at u. Note that P(U=u) = p&lt;sub&gt;U&lt;/sub&gt;(u).</td>
</tr>
<tr>
<td>F&lt;sub&gt;U&lt;/sub&gt;(t)</td>
<td>Cumulative pmf of a random variable U evaluated at t.</td>
</tr>
<tr>
<td>E[U]</td>
<td>The expected value of a random variable U.</td>
</tr>
<tr>
<td>ū</td>
<td>The sample mean of a given number of observations of the random variable U.</td>
</tr>
<tr>
<td>E[U</td>
<td>V=v] or E[U</td>
</tr>
<tr>
<td>∑ f(x)</td>
<td>“sum f(x) over all values of x”</td>
</tr>
<tr>
<td>∑ f(x)</td>
<td>“sum f(x) over all possible values of x, x ≤ y”</td>
</tr>
<tr>
<td>D(x) = E[Y</td>
<td>X=x]</td>
</tr>
<tr>
<td>d</td>
<td>Intercept for the linear regression function.</td>
</tr>
<tr>
<td>a</td>
<td>Slope for the linear regression function; a ≥ 0.</td>
</tr>
<tr>
<td>y&lt;sub&gt;i&lt;/sub&gt; = β&lt;sub&gt;0&lt;/sub&gt; + β&lt;sub&gt;1&lt;/sub&gt;x&lt;sub&gt;i&lt;/sub&gt; + ε&lt;sub&gt;i&lt;/sub&gt;</td>
<td>General linear model with 1 quantitative variable, β&lt;sub&gt;0&lt;/sub&gt; is the intercept, β&lt;sub&gt;1&lt;/sub&gt; is the slope, ε&lt;sub&gt;i&lt;/sub&gt; is a random variable (error term) with mean 0.</td>
</tr>
<tr>
<td>σ&lt;sup&gt;2&lt;/sup&gt;</td>
<td>The variance of ε&lt;sub&gt;i&lt;/sub&gt; and y&lt;sub&gt;i&lt;/sub&gt;.</td>
</tr>
<tr>
<td>Ń , Ń , Ń , etc</td>
<td>An estimate of β&lt;sub&gt;0&lt;/sub&gt;, an estimate of β&lt;sub&gt;1&lt;/sub&gt;, an estimate of σ&lt;sup&gt;2&lt;/sup&gt;, etc.</td>
</tr>
</tbody>
</table>
LIST OF SYMBOLS AND ABBREVIATIONS (continued)

<table>
<thead>
<tr>
<th>SYMBOL / ABBREVIATION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_0, H_a$</td>
<td>Null hypothesis, alternate hypothesis.</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Level of significance used in a confidence interval or hypothesis test.</td>
</tr>
<tr>
<td>$r^2$</td>
<td>Coefficient of determination in least squares regression.</td>
</tr>
<tr>
<td>$df$</td>
<td>“degrees of freedom”</td>
</tr>
<tr>
<td>$SS(\text{Res})$</td>
<td>“residual sum of squares”</td>
</tr>
<tr>
<td>$Q$</td>
<td>Difference in residual sums of squares from two least squares regression model fits.</td>
</tr>
<tr>
<td>$t_{\alpha,v}$</td>
<td>$\alpha$ percentile of the student’s t distribution with $v$ degrees of freedom.</td>
</tr>
<tr>
<td>$t_{\text{calc}}$</td>
<td>A calculated t statistic.</td>
</tr>
<tr>
<td>$F_{\alpha,v_1,v_2}$</td>
<td>$\alpha$ percentile of the F distribution with $v_1$ numerator and $v_2$ denominator degrees of freedom.</td>
</tr>
<tr>
<td>$F_{\text{calc}}$</td>
<td>A calculated F statistic.</td>
</tr>
<tr>
<td>Cook’s $D_i$</td>
<td>A statistic used to measure potential influence of the $i^{th}$ observation in least squares regression.</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

Schedule estimation lies at the heart of every software development project. Scheduling and planning play a significant role in successfully delivering software products on time and within budget. Empirical and theoretical schedule estimation models for software development (e.g., COCOMO [Boehm, 1981]) can be applied during the initial stages of a project. These models operate on initial estimates of the product attributes (e.g., lines of code), so the accuracy of the schedules they produce varies with the precision of the estimates. The interpretation of the term “accuracy” may become unclear when one considers the fact that different schedule estimates can produce different project behavior [Abdel-Hamid-Madnick, 1986]. For example, if the time needed to develop a software product is overestimated, then the project may progress more slowly than if it had been underestimated. Other factors, like the availability of resources, also affect project progress.

This study investigates relationships between the time a software component begins a lifecycle phase and the time it takes to complete that phase. A tradeoff exists between the product quality, size, and its other qualities, and the schedule under which it is developed. For example, the product influences the schedule when it cannot be developed in the time allotted: the schedule must be pushed back (or “slipped”) to allow for the additional time needed for the product development. Conversely, the forecasted schedule can affect the development time of the product. If more time than required is scheduled, developers can use the excess time for activities unrelated to the project, but still finish their work on time. This is a consequence of Parkinson’s Law, which states, “Work expands so as to fill the time available for its completion” [Parkinson, 1957]. Although “Parkinsonian estimation” [see Boehm, 1981] is not recommended for estimating schedules, we may still utilize the
effect to partially describe project behavior after a schedule has been set. This study examines the impact of the schedule on the project itself. Some other studies have also investigated this concept quantitatively using simulations (e.g., [Abdel-Hamid-Madnick, 1983, 1986]).

Software engineering, as it is practiced today, is largely a problem-solving and social activity; many people communicate and cooperate to design, implement and test software systems to be used by people. The dynamics of social interactions are hard to predict. The relationships between people, and with the work they perform, create the difficulties associated with estimating a software project schedule. The dynamics include "feedback loops", resource utilization, productivity, and so on. Feedback loops occur when events in the present have a direct impact on future work. An example is design rework due to a design flaw discovered during coding. A broader example is future schedule estimation based on past project behavior (or use of "historic" prediction models). Resource utilization and productivity can be directly linked to individual software developers. Predicting how individuals will, or should, utilize resources and predicting and measuring their productivity have always been difficult in software projects. Take, for example, a statement from [Abdel-Hamid-Madnick, 1983], due to Farquhar:

"Unable to estimate accurately, the manager can know with certainty neither what resources to commit to an effort nor, in retrospect, how well these resources were used. The lack of a firm foundation for these two judgements can reduce programming management to a random process in that positive control is next to impossible."

Process control depends on accurate and timely information concerning the progress of the project. With lack of this kind of information comes randomization. The approach taken in this study assumes that the exact time spent developing any given component of a software system is not deterministic. The development times of product components, and therefore
of the entire product, are related to the schedule that was forecast at the beginning of the project. The impact of schedule changes, which are mostly caused by the status of the project's completion, are also considered. Although it is assumed that the development time of system components cannot be known to a high degree of accuracy in advance, we hope that the behavior of the project as a whole can be described. This is sometimes referred to as macro-modeling. Some schedule estimating models use this approach (e.g., Putnam's model (referred to as "SLIM") [Putnam, 1978] or [Putnam-Fitzsimmons, 1979] and Boehm's COCOMO [Boehm, 1981]).

1.1 Goals of the Study

The main goal of this work is to provide a description, in the form of a regression model, of the behavior of software projects developed under a certain process. The main features of the projects and the process include:

- The process is partly driven by milestone dates (e.g., a date set for product release) which delimit phases in the entire development schedule.

- The product component life-cycle phase durations behave as response variables. One predictor variable is the time available to a planned milestone date for each component.

- The process is applied in overlapping projects that correspond to successive releases of the product with enhancements.

Other objectives of this study are

- To provide a description of the product component completion behavior for software project phases. In this context, any given software product is comprised of smaller objects, called deliverables (or components), which include programs and documentation (i.e., software).
• To describe software project behavior by utilizing relationships between life-cycle phase starting times and phase durations for components, as well as placement of milestone dates.

• To show that the model is self-consistent and that the specific environment it was developed to describe conforms to the model assumptions. Simulation of project behavior is used to show that the model describes component finish time distribution parameters adequately. The model assumptions are checked using empirical data, and the model parameters are estimated from the data.

• Develop the model in such a way as to facilitate expansion and "programmability". Programmability refers to the ease of implementation and tailoring of the model in a computerized software process modeling system.

Discussion Achieving the first objective, describing product component completion behavior for software project phases, should allow managers and software developers to better predict the completion time of their project for any life-cycle phase. Such predictions could be used to analyze risks associated with not meeting planned project deadlines. Achieving the second objective, describing project behavior by utilizing product, process and schedule information, should foster interest in software process modeling, and emphasize the fact that one cannot separate the three when describing software project behavior.

The third objective, model evaluation, aims at showing that the model adequately describes observed project behavior. Simulations also serve as a means of investigating effects that cannot be easily expressed analytically. Achieving the fourth objective, facilitating model expansion and programmability, helps ensure that the model paradigms will be investigated further, and can be adapted and instituted in different environments.

† Throughout the report, "start time" refers to the time when a product component enters a life-cycle phase. Similarly, "duration" refers to the time taken to finish the phase.
1.2 Model Motivation

To describe a development process, we need to discover any relationships that exist among the attributes of components that constitute the product. By analyzing data from past projects, it is possible to understand the relationships that exist in a project. One interesting relationship was observed between the start time and the duration of components' development in all software projects observed. The phase discussed spans the period from the program high-level design to the completion of the coding, unit testing and integration. Similar relationships were observed for other life-cycle phases, which are detailed in Section 2.2.

Figure 1-1 shows a scatterplot of data from a past project†, which we shall call “Project 1”. It indicates a near-linear relationship between component start time and duration. On the graph, and in subsequent analyses, program start time is measured with respect to a planned milestone date (0 on the horizontal axis) for a specific life-cycle phase. As mentioned before, the milestone in this case is the completion of coding, unit testing and integration. In this diagram, two different kinds of components were identified in the project: 1) those that are new to this project, and 2) those that have been inherited, or carried over, from a past project (or past product release) and are being incorporated into the current release. It was determined that this distinction is an important one in this model. Note also that the carried-over components appear to be separated into two sub-groups. The groups can be identified in Figure 1-1 as the two parallel bands of the ‘/’ symbol. Similar relationships have also been observed in four other projects.

† Data used by permission. The scale appearing on the axes on all graphs and any product and date-related information have been altered to provide discretion.
Figure 1-1 Scatterplot of component duration vs. start time for components in Project 1. The phase shown spans high-level design to the completion of coding, unit testing and integration.

Using the method of least squares, linear regression functions were fit to the data shown above, one for each type of component: new and carried over. The regression analyses revealed that the relationship between the component start time, relative to the planned milestone date, and the component development duration is described well by a linear regression model. Further analysis shows that the separation of the components into "new" and "carry-overs" can be statistically justified (see Section 3.2). These observations, and others like them, led to the development of the theoretical description of the underlying process given in Section 2.

Histogrames of the start and finish times for components in Project 1 are shown in Figure 1-2. Both distributions of new components have approximately normal, or Gaussian, shapes. Similar shapes were observed for all other projects on which data was available.
Figure 1-2  Histogram of start time (a) and finish time (b) for components in Project 1.

A cumulative frequency diagram of the finish times of components in Project 1 is shown in Figure 1-3. This device is useful for prediction of the way project components will finish. In the figure, the "sum" curve represents the total number of components finishing at a given time. Estimates of the probabilities can be made from the cumulative frequency
distribution. Of interest is the probability that a certain percentage of product components will finish by a certain time. The model described in this study can be used to project and estimate the cumulative frequency curve.

![Cumulative Frequency Plot](image)

**Figure 1-3** Cumulative frequency plot for component finish times for components in Project 1. The components are partitioned according to their classification as new or carried over, as discussed earlier.

It was determined that a linear, additive regression model provides an adequate description of the component phase durations (and therefore finish) behavior. In this case, the model used component start time as the predictor variable and component phase duration as the response variable. In general, regression provides a model where the mean level of the response variable can be estimated as a function of one or more predictor variables. The level of confidence depends on the variance of the response variable about its expected value, given by a *regression function*. Also necessary for confidence interval statements and hypothesis testing is the normality of the random error term in the regression model. Using regression diagnostics, it was determined that this assumption is not grossly invalid in the observed environment.
For reasons of simplicity, the predictor variable selection was limited to a single quantitative variable: component phase starting time. Other quantitative independent variables may prove significant in other environments, although not necessarily in a linear and/or additive fashion. A measure of lines of code could be used as an additional quantitative variable. Also, qualitative variables may prove useful in reducing variability in the observed dependent variable. Qualitative variables can be incorporated into regression models in the form of class variables (e.g., [Rawlings, 1988], esp. chapter 8). Managers, departments and program type are examples of possible qualitative variables.

1.3 Review of Past Work

This section briefly discusses some existing software schedule estimation models. The concepts behind Parkinson's Law and the Deadline Effect, which are important to this model, are defined and discussed.

1.3.1 Software Schedule Estimation Models

Myers [Myers, 1978] points out that there exist basically two ways of estimating software costs and schedules: micro-estimation and macro-estimation. To apply micro-estimation, one must distinguish all of the activities which are required to finish the final product. Every single activity gets a cost and schedule estimate. The activities can be arranged in a network, mainly to exploit parallelism among the activities (see [Boehm, 1981] or [Hillier-Lieberman, 1986]). This is the basis of the PERT and CPM approaches. Critical paths determined from the network can be used as a lower bound on the total time needed. By definition, any slippages that affect activities on the critical path must force a slippage of the entire project. The main difficulty in the micro approach stems from inaccurate estimates of the individual activity attributes. This difficulty becomes more unmanageable as more activities are added along with more people.
On the other hand, macro-estimation is most applicable to large systems. Estimates are made at the *product* level, and not at the individual component level. Initial estimates for the product are used early in the project life-cycle in order to predict the manloading (number of staff required at any given time) and the schedule. An example of a macro-estimation model for software systems is Lawrence Putnam's SLIM [Putnam-Fitzsimmons, 1979]. The form of the Putnam model is

\[ S_s = C_k K^{1/3} t_d^{4/3} \]

where \( S_s \) is the number of lines of delivered source code, \( C_k \) is a constant (called the "state of technology" constant), \( t_d \) is the development time in years, and \( K \) is the life cycle effort in man-years. As can be seen, the model relates the size of the product to the effort and time required to develop it. The effort and the development time can be related more generally by

\[ \text{Effort} = c / t_d^4 \]

where Effort is an estimate of effort in man-years, and \( c \) is a constant in the range 14 to 15 [Boehm, 1981].

The COCOMO model [Boehm, 1981] is another example of a macro-estimation model. It relates the size of the product to the required effort and the development time. The COCOMO model is an example of a more general set of multiplicative models. This type of model estimates the effort with equations of the form

\[ \text{Effort} = a X^b \]

where \( a \) and \( b \) are constants, and \( X \) is a measure of product size. The models estimate development time with equations of the form

\[ \text{Time} = c(\text{Effort})^d \]
where \( c \) and \( d \) are constants, and Effort is an estimate of the effort. Surveys of several models of this type can be found in [Boehm, 1981] or [Fairley, 1985].

The COCOMO model is flexible because it can utilize added information and become a micro-estimation tool. COCOMO can be thought of as a micro-estimation tool in its intermediate and detailed forms.

Macro models like the Putnam and basic COCOMO models are macro in the sense of the product and the time interval considered. Product-level estimates are used to estimate the nominal effort and calendar time (schedule) required to develop the product. However, COCOMO can be used as a micro-model, to “focus in” on both the product and the time intervals. A shortcoming of macro-estimation techniques, as explained by Boehm, is that they assume that the (cost) driving factors are applied equally throughout the entire lifecycle and the product (e.g., that all the personnel are equally qualified and motivated, the product components are of similar complexity, and so on).

1.3.2 Parkinson’s Law and the Deadline Effect

Software developers spend their time on many things, and they divide it up among the activities they must perform. Suppose a developer (or a team of developers) reports he started an activity of interest, \( A \), at time 0 and finished at time 5. However, what he fails to report is that he worked on activity \( B \), which is unrelated to the completion of activity \( A \), from time 2 to time 3, at which time he was not working on activity \( A \). An activity duration of 5 units is reported, when only 4 units were needed to finish activity \( A \). The unpredictability of the partitioning of the developer’s time contributes to the randomness of reported times. This is likely to increase as the number of activities increases, particularly
if the activities "belong" to different projects. An illustration of a partitioning of total reported activity time appears below†.

\[\begin{align*}
\text{\(\square\)} &= \text{Time spent on other activities} \\
\text{\(\square\)} &= \text{Time spent on activity A}
\end{align*}\]

\textbf{Figure 1-4} Example of a partition of a software developer's time.

Parkinson's Law states that work will expand to fill the allocated time. This may occur because of an overestimate of the time needed for the project, or by activity time mismanagement on the part of the developers. Developer behavior could be affected by the knowledge of the planned milestone date. The programs that start (i.e., begin a life-cycle phase) closer to the planned milestone date take less time to complete than those that start further away. That is, the completion duration becomes more or less the amount of time that was originally available until the milestone date. Hence, it may happen that the \textit{same} program will require different amounts of time to finish a phase, depending on how close to the due date work on it commences.

\textit{I'm not sure the next couple weeks would be valid [for an effort study]. We are in our normal...panic to meet a key...milestone. This is not the norm and I think the study would not be valid based on the next couple weeks...}

The above comment is an excerpt from a reply to a call for volunteers to participate in an effort study. Several software managers were asked to participate in the study. The replies varied, but they carried two common themes: 1) "What's in it for us?" and 2) "This is the worst time for it". The first question is certainly valid, and one answer is: improvement in the planning and control of the software development process so that it is possible to bring a quality software product to market on schedule and within budget. The second theme is

† See Section 2.4 for more on how software developers spend their time.
intriguing. Why are the few weeks before a major milestone date so different from the rest, or "the worst time for it"? It's called the Deadline Effect:

*The amount of energy and effort devoted to an activity is strongly accelerated as one approaches the deadline for completing the activity.* [Boehm, 1981]

Sometimes software developers increase their effort closer to the milestone date in order to make the deadline. The Deadline Effect is loosely the converse of Parkinson's Law. Suppose a developer believes he cannot make a deadline by working at his present pace. He then puts in the extra effort (in the form of overtime, skipping lunch, etc.) to finish his activity by the due date. The time needed to complete the activity thus becomes the time that was allotted. Note that the extra effort the developer has put into the activity, by "borrowing" time from his other activities, is often not recorded in the reported development time.

As Parkinson's Law and the Deadline Effect suggest, the actual placement of milestone dates can have an impact on the time and effort devoted to activities (and therefore to their finish times). The impact, in some cases, causes planned milestone dates to become "self-fulfilling prophecies"†. Parkinson's Law and the Deadline Effect may be common to many software development environments, which shall be called *milestone date-driven*.

Based on the previous discussion, it could be assumed that not all components in a project finish early, even if the schedule is relaxed. Rather, a significant number of them may finish on time at best. The key point here is that a developer's perception of the time allotted partly governs the time and effort he puts forth to achieve a milestone. This effort, and therefore the development time of the components, depends on the placement of milestone dates. Of course, software managers are not free to arbitrarily place milestones

---

† See [Boehm, 1981] for more on Parkinson's Law, the Deadline Effect and self-fulfilling prophecies.
dates to speed up the overall process. For example, Boehm reports that schedules typically cannot be compressed to less than 75% of the nominal schedule [Boehm, 1981]. Milestone date placement requires a good understanding of the software product and the process, as well as a good prediction of how long the project and its individual components should take to complete. Any of the methods mentioned earlier (i.e., the Putnam or COCOMO models) can and should be used to estimate such quantities.

Simply scheduling the components of a project in a certain way could also lead to the self-fulfilling prophecy effect. For example, large programs will usually be given more time than the small ones, and so they will be started earlier, with the hope that they all will finish by the time the project is planned to be finished. We are not asserting that all programs will finish at the same time, on the due date, but rather that developers will tend to manage their time in such a way so as to finish as near to the due date as possible.

1.4 Scope of this Study

The focus of this study is on providing a regression model to describe the behavior of software projects developed under a phased, milestone date-driven software development process. The model is a stochastic macro-model. It describes behavior at the product level, but uses stochastic component-level information to arrive at the description. The basic assumption is that a schedule for the project is set, and that its planned milestone dates are known to all the staff working on the project. The planned milestone date of a life-cycle phase will be related to the duration of the components for that phase.

The model is a macro model at the product level, but it is a micro-model at the time level. The product is viewed as a collection of software components which behave in a stochastic manner. "Stochastic" refers to the unpredictable nature of the partitioning of developers’
time, and therefore to the time at which developers finish any given life-cycle phase. The model can be applied to any of the life-cycle phases, or to any combination of adjacent phases.

To illustrate the model, a linear, additive model is defined. Least squares regression is used to estimate the model parameters. For each product component, the expected phase duration is expressed as a function of one predictor variable: start time relative to the phase milestone date:

\[ \text{Duration} = a + bX \]

where \( X \) is the start time and \( a \) and \( b \) are constants. Other predictor variables could be identified and incorporated.

Parkinson's Law and the Deadline Effect will be quantified by assuming that the average time required to complete a phase will be the time that has been allotted. Once a schedule has been set, and a date chosen for the project completion, the model derived in this study can be used to describe the average behavior of the product components (provided they follow a development process similar to the one described in this study).

### 1.5 Overview of the Study

Section 2 covers the development and derivation of the model. Section 2.1 enumerates and discusses the definitions and assumptions of the model. A detailed description of the software environment is presented in Section 2.2. Section 2.3 contains the model development. Section 2.3.1 lays the groundwork for the model with notation and definitions. A description of the distribution of finish times is derived in Section 2.3.2. The expectations of program durations and finish times are discussed in Section 2.3.3. Section 2.3.4 discusses the implications of Parkinson's Law and the Deadline Effect, and
defines the model as a linear additive model. Formulas for estimating the variances of durations and finish times are given in Section 2.3.5. The model's implications with respect to schedule changes are considered in Section 2.4.

Section 3 contains an evaluation of the model. Both simulated and empirical data were used to evaluate the model. The model's performance on simulated component/project data in the absence of schedule changes is observed in Section 3.1.1. The effect of schedule changes on the project behavior is simulated in Section 3.1.2. The results of empirical data analyses are presented in Section 3.2.

Section 4 summarizes the study and its implications. The accomplishments and investigations of this study, and the shortcomings of the model are summarized in Section 4.1. Section 4.2 contains recommendations for future research.
2 MODEL DEVELOPMENT

The observations presented in Section 1.2 suggest that there is a relationship between the component starting time and its duration for a given life-cycle phase. It appears that the components that start closer to the planned milestone date for a phase will require less time to complete that phase than those that start further away. Although we cannot assert that observations from the present data guarantee a causal relationship, the dependence of program duration on its starting time relative to the milestone date seems reasonable. The dependence is suggested by Parkinson's Law and the Deadline Effect.

2.1 Model Definitions and Assumptions

Most terms used in this report have widely accepted meanings and are generally used in a broader context. More specific definitions are adopted in order to avoid ambiguity. The model can be expanded for use with broader definitions†.

DEFINITIONS

- A software product is a package of two or more deliverables. Products are sometimes called software systems.

- A deliverable is software: a program and/or supporting documentation. Deliverables were also called components in earlier discussions. A program is a collection of instructions that can be run on a computer. Throughout the report, however, program will be used as a synonym for deliverable or component.

- Every software project is governed by a process, which includes the full set of activities necessary to deliver the software product, including a particular life-cycle model.

† See [Boehm, 1981], [Brooks, 1975] and [Humphrey, 1988] for more on the definitions of software and the software engineering process.
• A software project defines the mapping between the software process and the software product. Every product is different, and requires a different mapping to the process.

• A milestone date is a point in time that serves as a goal for software product developers. The goal is the completion of certain deliverables, at which time the milestone is achieved.

• Milestones can be either hard or soft. The difference between the two types lies in the criteria required to achieve the milestone. The criteria required for a hard milestone are measurable and verifiable, like the completion of a detailed design document that has been approved by the developer’s manager(s) and peers. To achieve the milestone, the criteria are enforced. The criteria required for a soft milestone may be ambiguous, can lead to confusion, and usually do not entail verification of deliverables.

• A project’s life cycle is divided into phases. A program start time is the time at which the program begins a life-cycle phase. A program finish time is the time at which the program completes the same life-cycle phase. Both times will be measured with respect to a planned, fixed milestone date for that phase.

• A program phase duration, or just duration, is its finish time minus its start time.

In this report, all starting and finishing times for a life-cycle phase are scaled to preserve the time measurements across multiple projects. First, all the program start, finish times and project milestone dates are converted to an appropriate linear scale (e.g. Julian dates). The planned finish milestone date for the phase is subtracted from the starting and finishing times, and from the milestone dates. This scaling provides a distance to milestone date measure. Durations are measured in the same units as the scaled dates (e.g., days). The scaling technique also provides a basis for measuring the evolution of project schedules, since all dates are measured with respect to a single fixed date on the calendar time axis.
The definitions of start time and finish time apply to any life-cycle phase. For reasons of simplicity, the phase discussed in this paper is the design and coding phase. This phase is defined to begin at the start of high-level design and end at the completion of coding, unit testing and integration. The relationships observed for this phase have also been observed for other phases (e.g., specification to system test completion). The relationships tend to be stronger (i.e., show stronger correlation) when the phase milestone is a hard milestone. This is discussed in more detail below.

**ASSUMPTIONS**

1) *Program durations behave as random variables.*

This assumption allows the phase durations to be viewed as samples from populations that could be described by probability distributions. The assumption also permits the application of probability theory in order to make inferences. The information that is relied on the most, the reported start and finish times of a program development activity, is usually incomplete. That is, the duration reported for an activity is not necessarily an accurate measure of the time spent solely on that activity. The idea of the unpredictable nature of the time spent on activities was addressed earlier, in Section 1.3.2.

There may be a problem in the reporting of the times itself. If the developers are responsible for reporting their own times, they may report that a program has achieved a milestone when it has not, or they may finish the program, but not report that it is finished until the due date. The first kind of reporting error may occur because the developer interprets the status of his activity differently than his manager. The second type of error might be caused by simple negligence, or it might be partly caused by a misunderstanding of the value of accurate information.

Some of the problems linked to reporting can be associated with the milestones used in the schedule. If milestones are *hard*, or well-defined and measurable, there should be little disagreement between the managers and the developers about the status of an activity. The requirements for completion of a milestone are spelled out explicitly, and must be met unconditionally. If the milestones are *soft*, then opinions can determine whether the activity has reached the status or not. For this reason, soft milestones
should be avoided, or at least be kept off the critical path in the project schedule. Truly
not achieving soft milestones on time should not lead to a change in the final milestone
date, if it can be avoided [Brooks, 1975]. By definition, soft milestones are associated
with confusion, and one would like to be as definitive as possible about the finish date.

Note For the model, observations on the predictor variable (i.e., reported component
start time) are considered to be known constants. This means that reported data are
correct, and it is this reported data that will be used to build the model (see Section
2.2).

2) Program durations are independent of each other.

This assumption will always be violated somewhat in real software projects. For
example, product components are interdependent during integration and integration
testing. The assumption does, however, allow us to further simplify the derivation of
model formulas.

3) Expected program duration is, on average, equal to the time remaining
to the planned milestone date.

This assumption forms the basis of the linear modeling approach, which is used to
demonstrate the model. Expected program duration is assumed to decrease as program
starting time approaches the planned milestone date. This means that, on average,
programs that start closer to the planned milestone date will require less time to finish a
life-cycle phase than those that start further away. Variation in the duration for
programs starting at the same time will be caused by differing program attributes and
development conditions. These attributes could perhaps be incorporated into a more
general model (not necessarily as linear and/or additive functions).
4) **Deviations of program duration about their expected value are normally distributed with mean 0.**

It is also assumed

4a) **Deviations have common variance, \( \sigma^2 \), across all values of the predictor variable, and**

4b) **Deviations are independent.**

These are typical assumptions made in least squares regression. These assumptions do not seem to be artificial, based on regression diagnostics of historical data. The assumption also implies that the program durations are normally distributed about their expected value with common variance. Under the normality assumption, the least squares estimates are equal to the maximum likelihood estimates. For more on maximum likelihood estimation, see [Rawlings, 1988] or [Searle, 1971]. Assuming the deviations are normally distributed also provides the basis for hypothesis testing and confidence interval statements, but is not necessary to estimate by least squares.

**Discussion** Under the assumptions, which assume a process, the behavior of a software project can be described. The behavior is described in terms of the product components, the programs. During a phase, the development of a single program is an activity. The reported time required for an activity is considered to be a known instance of a random variable. The reported times include the time spent on other activities. The assumptions imply that developers will manage their time so that the activities they are responsible for take time approximately equal to the amount of time they are given. Variations in activity difficulty, individual developer ability, and the way developers manage their time cause the variations in activity durations. In other words, it is assumed that developers will "aim" for the due date (on average), and they will miss it with an error which is normally distributed.

It is also assumed that a program phase duration will depend on how "near" it starts to the milestone date for the phase. Under this assumption, it is not clear that a program's finish time, which is its start time plus duration, will also depend on the starting time. That is,
programs that start at very different times may have similar chances of finishing on time, but they will have very different durations. For this reason, duration is used as the dependent variable (rather than finish time). One could then focus on program attributes that affect duration for a given phase. Such attributes can possibly be used as variance-reduction factors. A measure of complexity serves as an example: if two programs start on the same day, and the first is quite simple and the second very complex, then they will likely take different amounts of time to design, implement and test.

2.2 Detailed Description of the Development Environment

An important part of the model development is the environment that contains the projects. This section elaborates on the attributes of the environment in which the model operates. The software development environment described here has practical value, in the sense that it, or its variants, is being used in industry on large software projects.

Software projects combine products and a process. The observed process life-cycle resembles the classic waterfall model (see [Boehm, 1981] or [Pressman, 1988]). The entire software development is divided into phases separated by milestones, or goals, to be attained at the end of each phase. Each program that makes up a software product follows the same life-cycle model, which is similar to that of the entire product. When a predetermined fraction of programs achieve a given milestone, then the product is said to have achieved that milestone. In order to deliver the completed product, all programs that remain in the product must achieve the final milestone. Any programs that do not must be abandoned, and perhaps re-instated at a later time, as part of another product release, during another project. It is not necessary that all program development starts at the same time. Also, it is not expected that all programs will achieve intermediate or the final milestones at the same time.
The integration and system testing is typically performed by a separate group of people from those who designed, unit tested and integrated the software. However, there is contact between the independent testers and designers. Recall that the modeling approach will be illustrated for the design and coding phase, which begins with high-level design and ends after coding, unit testing and integration.

The entire development life-cycle model is illustrated in more detail in Figure 2-1. The project proceeds from left to right, but it can backtrack, if needed. Reasons for backtracking (or feedback) include re-work on design because of an error that was discovered, or by a change in requirements. Backtracking is not limited to one phase.

Figure 2-1 The software development process. The entire interval is divided into smaller phases, which are labelled by the milestones that signify completion of the phase.
DESCRIPTION OF THE LIFE-CYCLE PHASES

Specification: During the specification portion of the development, the major commercial functions of the product are defined. Also, efforts are made to assess the feasibility, content, and size of the product. Project plans are created in this stage to define the progression of the process. The specification phase contains three subphases:

• Feasibility
  Program specification documents are written during the feasibility phase. In order to complete the phase, all required specification documents must be completed.

• Requirements
  The requirements phase focuses on planning development and finalizing requirements. Preliminary project schedules are drawn up, requirements are mapped onto development activities, and staffing is arranged. At this stage, the individual activities are usually (and should be) entered into an online database in order to track their development status.

• High-Level Design
  Detailed functional specifications are reviewed and finalized in this phase. Also finalized are development plans, including release priority, packaging, and schedules.

Design / Coding: Once the specifications are finalized, the development of the software to implement their functions is begun. The design / coding phase includes high-level and detailed design, and unit testing. It consists of two subphases:

• Detailed product design
  The design of the programs is performed in this phase. First, a high-level design breaks the program down into small modules and their architecture is developed. A detailed design of each module follows. To complete the phase, the high-level and detailed design documents must be completed, and each is reviewed before it is accepted.
• **Code, unit test and integration**

During this phase, program modules are implemented. Each programmer tests his modules with help from an independent tester. To complete the phase, the code must be inspected and tested by the designer, analyzed using code coverage tools, and placed under change control in a code library.

**Verification**: The verification phase focuses on regression and traffic testing (which simulates actual operational environments) of the existing software base, in order to ensure that it is not corrupted by any new software development.

• **Integration testing**

Established performance metrics related to failure rates must be satisfied before this phase is completed. The testing at this stage is at the product level (i.e., all the programs in the product have been coded, unit tested and integrated).

It is possible for more than one project to be underway at a time. Product developers divide their time among several projects. Each project applies the same life-cycle model (process), discussed above. The projects overlap in their different phases. For example, system testing in project N–1 may overlap design in project N and specification in project N+1. The goal of each project is to deliver an enhanced version of the software product, the next release. Work on the enhanced version of the software includes updates (including error corrections) to existing software as well as additions. Project planning may be simplified somewhat by the repetitive nature of the process and similarities in component functionality. An illustration of the overlapping nature of the projects is shown in Figure 2-2.
Figure 2-2  Illustration of overlapping projects in the environment.
2.3 The Single-Variable Model

In this Section, the model is expressed in terms of a regression function. A linear regression function is used to demonstrate the model.

2.3.1 Model Notation and Definitions

Capital letters are used to indicate random variables. For example: \( X \). Lower case letters indicate a specific realization, or instance, of a random variable. For example, \( X=x \) means "the random variable \( X \) has taken the value \( x \)". A random variable that is conditioned on another will be written as \( Y \mid X=x \), where \( Y \) is conditioned on \( X=x \). The notation "\( Y \mid X=x \)" or "\( Y \mid x \)" means "\( Y \), given \( X=x \)" , and is sometimes used, also. The distinction between the subscripted and non-subscripted variables is that the subscripted variables imply dependence. For example, \( Y \) denotes any random program duration, independent of start time; \( Y \mid X=x \) denotes a random duration for a given start time, \( X=x \).

**Definition** Let \( X \) be program start time\( ^\dagger \), \( Y \) be program duration and \( Z = X + Y \) be program finish time. Let \( Y \mid X=x \) be program duration for a program starting at time \( X=x \), and let \( Z \mid X=x \) be the program finish time for a program starting at time \( X=x \).

**Definition** Let \( X \) and \( Y \) be discrete random variables having joint probability mass function (pmf) \( p(x,y) \). The *conditional pmf* of \( Y \) given \( X \), is

\[
p_{Y \mid X}(y \mid x) = p(Y=y \mid X=x) = \frac{P(Y=y, X=x)}{P(X=x)} = \frac{p(x,y)}{p_X(x)}
\]

(2.1)

where \( p_X(x) \neq 0 \). [Trivedi, 1982]

\( ^\dagger \) Recall from Section 2.1 that "times" refers to reported times.
**Definition** Let \( D(x) = E[Z \mid X=x] \) be the regression function for duration on start time. \( D(x) \) is the expected duration for a program starting at time \( X=x \). The *theorem of total expectation* for any discrete random variables \( X \) and \( Y \) states

\[
E[Y] = \sum_x E[Y \mid X=x]p_x(x) = \sum_x D(x)p_x(x)
\]  

(2.2)

[Trivedi, 1982, esp. chapter 5]

This formula shows how to calculate the *unconditional* expectation of \( Y \), given the regression function, \( D(x) \) and the pmf for \( X \).

### 2.3.2 Calculating the Distribution of Finish Times

In this section, the start time distribution and the conditional duration distributions are related to the finish time distribution. Distributions of durations for a given starting time are represented by conditional distributions.

Using the definitions in Section 2.3.1, it follows that the pmf, \( p_z(z) \), for all program finish times, \( Z = X + Y \), can be derived as:

\[
P(Z=z) = \sum_{x=x}^z P(X = x, X+Y = z)
= \sum_{x=x}^z P(X = x, Y = z-x)
= \sum_{x=x}^z P_{Y \mid X}(z-x \mid x) P_X(x)
\]  

(2.3)

The last line follows from the definition of conditional pmf. Note that this distribution describes the probability that *any* program that starts will finish at time \( z \). It takes into consideration all possible pairs of \( x \) and \( y \) such that \( x + y = z \). For all pairs, we must have the finish time greater than the start time, or \( x \leq z \). This fact implies that \( y \) cannot be
negative, which fits the model nicely, but extra care must be taken when trying to apply the above formula to real situations. For example, when linear least squares regression is applied in Section 2.3.3, it may be the case that the model predicts a negative $y$ value (near the planned milestone date, 0, on Figure 2-3).

We can also calculate the probability that a program will finish at time $z$, given that it has begun at time $x_0$. The answer is simply $P_{Y|x}(Y=z-x_0|X=x_0)$. Given that a program starts at time $x_0$, there is only one value of $y$ that, when added to $x_0$, gives $z$. This value is, of course, $y = z-x_0$.

Let $p_Z(t)$ be the pmf of finish times, and let

$$F_Z(t) = \sum_{z \leq t} p_Z(z)$$ (2.4)

be the cumulative distribution of finish times. Suppose there are $N$ programs in the project. Then, the expected number of programs that will be finished by time $t_0$ is $N \cdot F_Z(t_0)$.

2.3.3 The Expected Values of Durations and Finish Times

In this section, the formula for the expectation for all program durations is derived. Recall that $Y = $ program duration. It follows by the theorem of total expectation (equation 2.2) that, for all programs, the expected duration is

$$E[Y] = \sum_x p_x(x) E[Y|x] = \sum_x p_x(x) D(x)$$ (2.5)

In the case where $D(x)$ is a linear function of $x$, equation 2.5 can be reduced to $E[Y] = D(E[X])$.

For a given starting time, $x_0$, the conditional expectation of finish time, $Z_{x_0}$, is given by
\[ E[Z_{x_0}] = E[x_0 + Y_{x_0}] = E[x_0] + E[Y_{x_0}] = x_0 + D(x_0) \] (2.6)

Using the rules of expected value and Equation 2.5 above, we can derive the expected finish time \( Z = X + Y \), of all programs by


Equation 2.7 can also be derived using the theorem of total expectation, by summing over all possible values of \( x \), the expression for \( E[Z_x] \) times \( p_x(x) \). In the case where \( D(x) \) is a linear function of \( x \), equation 2.7 can be reduced to \( E[Z] = E[X] + D(E[X]) \).

### 2.3.4 Applying Least Squares Linear Regression

The assumptions listed in Section 2.1 suggest least squares linear regression can be applied to estimate the parameters in the relationship between program start time relative to the planned milestone date and duration.

Suppose the relationship between starting time and expected duration be linear, as is suggested by the observations presented in Section 1.2. Then, we can expect a relationship like the one shown in Figure 2-3. Let \( D(x) = d - ax \) be the expected duration for a program starting at time \( x \), which is a line with slope \(-a\) and intercept \( d \), where \( a \geq 0 \).
Observe that start times have been emphasized as being negative. The definition of start time requires the subtraction of a program’s start time and the planned milestone date (see Section 2.1).

Figure 2-3 clearly indicates that a program that starts closer to the planned milestone date will require less time to finish than one that starts further away from the milestone. Of course, this behavior cannot be expected to carry on right up to the milestone date in reality, since all programs have some certain minimum development time. However, for programs that start very close to, or beyond the planned milestone, it will be assumed that the linear relationship holds, owing to the Deadline Effect. This should rarely be observed, since programs are usually started before the planned milestone date.

A distribution of durations, with expected value $D(x)$, is associated with any given program start time. That is, if two or more programs start at the same time, they most likely will not all take exactly the same amount of time to finish. The distribution of durations for each starting time is caused by differing program attributes and other program-related metrics,
including the developer and complexity. On the basis of historical data analysis, probability distributions could be assigned to suit a particular development environment. For this model, assumptions have been made about the duration distributions for a fixed start time in order to conform to least squares regression model (i.e., for hypothesis testing and confidence interval statements). Figure 2-3 also illustrates the idea of the consideration of a distribution of durations for a given start time. These distributions can thought of as conditional distributions, with the conditioning being on start time.

**The Slope of the Regression Line**

Assuming the linear relationship exists, what is its expected behavior? Should it have slope \(-2\), or \(-1/2\), and what should the intercept be? Suppose the work on a given program expanded to fill the allotted time, or that it was accelerated to finish on time (assumption 3, Section 2.1). Then, on average, the development time would be equal to the allotted time. Therefore, the slope of the line in Figure 2-3 should be \(-1\), since the time needed for development (duration) is the time allotted (start time, relative to the planned milestone date).

If the planned milestone date must be changed to a new date for any reason, the model changes. This illustrates the proposed dependence of program development time on milestone placement. The developers working on programs that have not finished by the time the milestone date is changed are now targeting a new date. The idea of schedule changes is discussed further in section 2.4, and is simulated in Section 3.1.2.

Let

\[ y_i = \beta_0 + \beta_1 x_i + \epsilon_i \quad (2.8) \]

where \( \epsilon_i \) is a random error term with mean 0.
This is a common definition of a linear regression model with one independent quantitative variable. $\beta_0$ is the intercept term and $\beta_1$ is the slope of the regression line. $y_i$ is an observation on the dependent variable, phase duration, and $x_i$ is an observation on the independent variable, phase start time.

The method of least squares gives the best linear unbiased estimators of $\beta_0$ and $\beta_1$. They are best in the sense that they have minimum variance among all estimators. If $\hat{\beta}_0$ and $\hat{\beta}_1$ are estimates for $\beta_0$ and $\beta_1$, respectively, then $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ is an estimator for $E[y_i] = \beta_0 + \beta_1 x_i$ [Rawlings, 1988].

The interpretation of the line, from Figure 2-3, is that $D(x) = E[y|X=x]$, or that for a given starting time, $x$, the expected duration is $D(x)$. The conditional expected value of a program duration is thus a linear function of the expected value of the start time associated with it. $D(x)$ is the regression function of duration on start time. We therefore use $\hat{\beta}_0$ to estimate $d$, the intercept. Similarly, we use $\hat{\beta}_1$ to estimate $a$. Note that $a \geq 0$, so $-\hat{\beta}_1$ will estimate $a$.

Assumption 4 in Section 2.1 states that the random errors, $\varepsilon_i$, are normally distributed with equal variance. This assumption provides the basis for confidence interval statements and the validity of hypotheses testing. It also implies that $\hat{\beta}_0$ and $\hat{\beta}_1$ are normally distributed [Rawlings, 1988]. The applicability of these assumptions in our setting is addressed in Section 3.2. A summary of confidence intervals in linear regression appears in the appendix.
2.3.5 Estimating the Variances of Duration and Finish Times

Under the assumption that the random errors, $e_i$, are normally distributed with equal variance, $Y_1, \ldots, Y_N$ also have independent, normal distributions\(^\dagger\). Furthermore, the true variance of duration, $y_{x_0}$, for a given start time of $x_0$, is equal to the variance of the error, $\sigma^2$. Let $\hat{\sigma}^2$ be the mean square error from the fit.

For a given value of the independent variable, $x_0$, the estimated variance of the mean response in the dependent variable is

$$\hat{\sigma}^2 \left[ \frac{1}{N} + \frac{(x_0 - \bar{x})^2}{\sum_{i=1}^{N} (x_i - \bar{x})^2} \right]$$

(2.9)

Also, for a value of the independent variable, $x_{new}$, the variance of the prediction of dependent variable, $y_{pred}$, is

$$\hat{\sigma}^2 \left[ 1 + \frac{1}{N} + \frac{(x_{new} - \bar{x})^2}{\sum_{i=1}^{N} (x_i - \bar{x})^2} \right]$$

(2.10)

Recall that the probability that a program will finish at time $z$, given that it has started at time $x_0$, is $P_{Y|X}(Y=z|x_0|X=x_0)$. It follows that the variance of the estimated expected finish time, for a fixed starting time $x_0$ is given by equation 2.9. Similarly, the variance of predicted finish time for a fixed starting time $x_0$ is given by equation 2.10 above.

\(^\dagger\) Here, suppose N points of the form $(x_i, y_i)$ are available in the data set. See the appendix for further discussion on linear regression.
2.4 The Effect of Schedule Changes

Schedule "slippage" dominates the history of software development project schedules. The slippage is generally caused by a tendency to underestimate the time needed to complete a project. Some of the reasons for the underestimates are listed in [Fairley, 1985]. Fairley summarized a Bell Labs time and motion study of 70 programmers conducted by Bairdain in 1964. The study revealed how programmers typically spend their time. Surprisingly, only 13 percent of programmers’ time was spent on writing programs! The rest of their time was spent on:

- Reading programs and manuals 16%
- Job communication 32%
- Personal 13%
- Miscellaneous 15%
- Training 6%
- Mail 5%

(Bell Labs, 1964)

Therefore, Fairley reports that a major reason for underestimating software project schedules is failing to account for the $13 + 15 + 6 + 5 = 39$ percent overhead time for the programmers and the $16 + 32 = 48$ percent overhead time for job communication and reading of manuals and programs. Such schedule underestimates will lead to schedule changes.

Only schedule changes that require a change in the milestone date will be addressed. Thus, schedule change will mean a change in the finish milestone date. Schedule changes do not necessarily require a change in the modeling approach. A change in a schedule only affects the progress on programs that remain unfinished at the time the change is announced. It will be assumed that the developers view the new due date (milestone date) as they viewed the previous one. More on this can be found in section 3.1.2, where a simulation is run with this approach.
When a schedule slip is announced, how do the developers react? Obviously, the progress on programs that have already finished by the time of the announcement (what we call the *slip date*) will not be affected. For any programs that have not yet begun by the slip date, one could assume that their developers view the new milestone date as they would view any other. That is, the developers will tend to take the allotted time and finish the program near to the (new) due date. So, the main focus is on those programs that began prior to the slip date and did not finish by that time. One could argue that developers will fill their remaining time (i.e., the time from the slip date to the new milestone), too, for reasons similar to why they fill the entire interval. Other reasons for this behavior include:

- The developer could never have finished by the old due date, and he needs the extra time to finish. This reason, if it is widespread, may have actually motivated the decision to slip the schedule.

- The developer could use the extra time to fulfill the specifications more completely.

- The developer could fill the extra time by adding "bells and whistles" to the program.

- The developer may spend more time performing unit testing.

- The developer might use the extra time to work on "slack time" activities, like reading mail, training, personal activities, or working on another project. A more detailed account is given in [Boehm, 1981].

Some of these possibilities were addressed before, when Parkinson’s Law and the Deadline Effect were discussed. In this report, it is assumed that the developers will indeed fill the extra time:

If a k-unit schedule slip occurs at time t, expected phase durations of programs increases by k units.
It is difficult, however, to know if those programs that were originally going to be finished just after the original planned milestone date will in fact become later, or if they will maintain their original "course". With the assumption above, it follows that they will tend to finish later on average (because of the slip), but whether or not this true in reality is debatable. It may not be generally known which programs would have been finished just after the milestone date in a real project, anyway. No conclusive empirical data on this behavior could be obtained.

Note that when a schedule slip is announced, the programs in the project are then partitioned into two sets: 1) those that have already finished and 2) those that have not finished. Some restrictions on the unfinished programs are enumerated. New durations are re-assigned to them, based on the new milestone date. According to the model and the assumption stated above, if the schedule is slipped k weeks, then the expected durations of the unfinished programs increases by k weeks. This, in effect, is the same as if the work on the program had begun k weeks earlier, according to the model. It is quite clear that if the program had not finished by the slip date, then it cannot finish by the slip date under the new milestone conditions either.

Before a schedule slip is announced, it is assumed that the program durations would be normally distributed around their expected values with equal variance. When the slip is announced, and the unfinished programs are re-assigned durations, another question is what happens to this variance? It could be that it remains the same. The equations derived in Section 2.3.4 imply that a change in the variance of durations will necessarily change the variance of the finish times. The relative size of the slip and the number of unfinished programs at the time of the slip announcement will impact the variance. Determining what actually happens to the variance has been difficult to ascertain from historical data.
The approach of re-applying the model using the new planned finish date is illustrated below. It does have limitations, however. For example, when applying the model only to those programs that are unfinished, a linear constraint is applied to the durations and start times (see above). In other words, it must be the case that start time plus duration is greater than the slip date. It is clear, then, that the durations cannot always be normally distributed about their expected values. This is also a shortcoming of the linear model approach, in general. This effect will occur when the schedule change is small with respect to the variance of the random error.

![Figure 2-4 Illustration of re-applying the model after a schedule change.](image)

There are, of course, other ways of handling the programs that remain unfinished by the slip date. Suppose a schedule is slipped \( k \) weeks at time \( t \). If a program has not finished by time \( t \), the model could be re-applied, using this new time to milestone as a start time, to obtain a duration for the time remaining (from \( t \) to the new finish milestone date). The new duration would then be added to the amount of time already spent on the program. Since all durations are greater than or equal to 0, the constraint of having new finish time greater than the slip date would automatically be satisfied.
Both approaches can be rather pessimistic in the sense that it is possible that few programs will finish near the slip date. The reason is that the expected finish time for the uncompleted programs becomes the new milestone date, and so many of the finish times will be near it. If the slip is large enough with respect to the variance of finish times, then not many finish times will fall near the slip date. It may be the case that in real projects, programs that would have been finished just after the slip date will finish as they would have if no slip had been announced (since they were close to completion, anyway).

Again, other assumptions could be made about the variance of the unfinished programs to solve this problem. More empirical data analysis is needed, however, before a definitive statement can be made. Evidence could not be found in abundance, so the problem must remain beyond the scope of this study. The implications of the first assumption above are investigated in section 3.1.2.
3 MODEL EVALUATION

In this section, the model is evaluated using simulation and empirical data. The goals of the simulation studies are to

- Validate the form of the model-generated finish time distributions. This is done by simulating start times and applying the model assumptions for the linear regression example. The simulation shows that the model adequately describes general and specific project behavior.

- Explore implications of schedule slippages. Describing the effect of a schedule slip analytically is beyond the scope of this study. However, simulation plays an important role in investigating the applicability of certain assumptions (see Section 2.4) without having to describe the behavior analytically.

The empirical data are used to

- validate the slope parameter assumptions used in the model. Recall that the expected value of slope is \(-1\); and

- verify that the model's assumptions (see Section 2.1) are satisfied by the data. We are mostly concerned with the assumption of the applicability of the linear model.

3.1 Model Performance on Simulated Data

The first evaluation of the single-variable model is based on simulated data, using the linear regression example. The simulated data include program starting times and the random errors associated with the linear model presented earlier. The linear model considers the expected duration of a program, for a given starting time, and a random error. Using simulation, it is shown that the model adequately predicts the distribution parameters derived in Section 2.2. Analysis of finish time distributions shows how project components finish, given that the initial assumptions (see Section 2.1) are satisfied.
3.1.1 A Simulation with No Schedule Changes

Distributions of starting times are simulated by a sample from a normal population. Motivation for this choice comes from observations of historical data. Figure 3-1 shows a histogram of program starting times from Project 1. The bulk of the times appear to be approximately normally distributed.

![Histogram of actual program start times in Project 1.](image)

**Figure 3-1** Histogram of actual program start times in Project 1.

In accordance with the model, assume that the programs’ durations have expectation equal to the amount of time they are allotted (i.e., their “distance” to the planned milestone date). A random error term is simulated by a random normal deviate, with mean zero and given variance. The variance size employed is similar to the mean square error from an actual regression of program duration on start time from similar historical data. The variance is the same for all values of start time.

† Only new programs are shown. See Section 1.2 or 3.2 for a discussion on program types.
Table 3-1 Summary of input parameters for the simulation.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>Variance</th>
<th>N (sample size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Time</td>
<td>-30</td>
<td>36</td>
<td>250</td>
</tr>
<tr>
<td>Random Error</td>
<td>0</td>
<td>20.25</td>
<td></td>
</tr>
</tbody>
</table>

Start times and durations were simulated using the above parameters. The sample size was \( N = 250 \). Say that a simulated program consists of a simulated start time, and its corresponding simulated duration and finish time. For all of the simulated programs, the finish time was calculated as start time plus duration. A histogram of the simulated program finish times is shown in Figure 3-2. The corresponding cumulative frequency diagram is shown in Figure 3-3.

![Histogram of simulated program finish times.](image-url)
Figure 3-3  Cumulative frequency plot of simulated program finish times.

Figure 3-4 shows a histogram of actual finish times from Project 1 (new programs only). Notice that it resembles the histogram of simulated finish times in Figure 3-2. Figure 3-3 shows the cumulative frequency plot of simulated program finish times. This curve also mimics the behavior of the actual cumulative distribution, shown in Figure 3-5, quite well.

Figure 3-4  Histogram of actual program finish times in Project 1.
Figure 3-5  Cumulative frequency plot of actual program finish times in Project 1.

Discussion  The results of the simulation indicate that the behavior of the project can be described in terms of a regression model. In this case, the regression function was linear. Treating the program durations as random variables gave results consistent with those observed. The simulation is readily programmable, and so it could be integrated into a quality management/process modeling system. It could be used to gain initial estimates of project behavior.

3.1.2 Simulating Effects of a Schedule Change

This section briefly discusses some possible impacts of schedule changes on the behavior of the project. The discussion is limited to schedule "slips", which involve a lengthening of the schedule.

Table 3-2  Summary of input parameters for the schedule slip simulation.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Unslipped Mean</th>
<th>Unslipped Variance</th>
<th>Slipped Mean</th>
<th>Slipped Variance</th>
<th>N (sample size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Time</td>
<td>-25</td>
<td>36</td>
<td>—</td>
<td>—</td>
<td>300</td>
</tr>
<tr>
<td>Random Error</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
The starting time distribution for this simulation was similar to the one used in the previous section. A schedule slip of 5 time units was “announced” at time 0 (the slip date), which is also the planned milestone date. One data set was created in the same way as in the previous section to mimic behavior for an unslipped schedule. A second data set was created for the slipped schedule. The simulated programs that remained unfinished at time 0 were re-assigned durations according to the following rules. The expected finish time for all unfinished programs increases by 5 units, because the slip is 5 units. Because the expected duration, according to the model, is the negative of the start time, it is as if the unfinished programs had actually started 5 units earlier than they did. The durations were assigned according to this rule, but the variance about their expected value was changed. The new value used was 1/4 of the original variance, which reduces the standard deviation by 1/2. This choice is based on historical project observations.

A histogram of the simulated program finish times for the first data set appears in Figure 3-6. Figure 3-7 shows the distribution of finish times for the programs on the slipped schedule. The programs that were not finished by time 0 are shaded in both diagrams.
Figure 3-6  Histogram of simulated program finish times, with no slip in the schedule. The shaded area indicates the programs that were not finished by the slip date (0).

Figure 3-7  Histogram of simulated program finish times, with a 5 unit slip in the schedule announced at time 0. The shaded area indicates those programs that were not finished by the slip date (0).
It is interesting to note that 22 programs finished after time 5 in the unslipped schedule simulation, whereas 79 programs finished after time 5 in the slipped schedule simulation. Also, a total of 143 programs did not finish by the slip date, 0, in both simulations; but, in the unslipped schedule simulation, their mean was 3, whereas in the slipped schedule simulation, their mean was 5.2. Obviously, in this case, schedule slippage was more detrimental than beneficial. For this simulation, the slip announcement actually made 57 of 143 programs later than they would have been under the unslipped schedule. These numbers may indicate that the simulation is too simplistic. It may be that outside pressures on developers will inspire them not to miss a slipped deadline in such numbers.

This simulation, and others, indicate that slipping a schedule may make the project later. It might be better to not announce the slip at all, but to hold the developers to the current schedule instead, with the hope that they will put in extra effort to minimize their lateness. This might force the Deadline Effect to compensate for the actual schedule slippage. It is highly questionable whether this holds in real projects. At the time this report was written, only limited empirical data existed on schedule slips and on what developers do with their extra time when they know a schedule has been relaxed. More data and probably a more elaborate simulation of developer behavior are required before a meaningful determination can be made.
3.2 Using the Model to Describe Historical Data

3.2.1 Discussion of the Historical Data

Throughout this discussion, the phase that is discussed is the design and coding phase, which is defined to include high-level design to the completion of coding, unit testing and integration. So, start and finish times for programs refer to this phase only. The historical data indicate that the projects in the environment contain two distinct classes of programs:

1) programs new to the project, and
2) programs carried over from a past project.

The new programs are those that are begun and delivered (or released) solely in the current project. The carry-over programs are programs on which work was begun during another project, but whose release was delayed until the current project. It could not be determined with confidence what proportion of work was done during the past and current projects.

Recall that the observed process is repeated in overlapping intervals (see Section 2.1). Work on carry-over programs does not always begin in the previous project. It is possible for work on some carry-over programs to have begun two or more projects in the past. Discussions with company employees indicated that the carry-over programs are of two distinct classes themselves:

a) those that were to be developed and finished in a past project, but whose release was delayed, and

b) those that were begun in a past project, but were not intended to be released in that project; they were always intended to be released in the current project.

This distinction is an important one to make in this model, since the type of carry-over program will dictate which finish milestone (past or current) at which it was “aiming”. The data clearly indicate the new/carry-over division. Unfortunately, the data does not clearly indicate the division of the carry-over programs. One possible way of separating the carry-
over programs depends on the finish dates declared in the past project. If the program was labelled as a carry-over, and its finish time is less than or equal to the finish date of the past project, then it is likely that it is type a) carry-over described above; otherwise, it will be of type b).

The carry-over programs tend to have significantly different mean finish time than the new programs. As an example, a histogram of program finish times for Project 1 is shown in Figure 3-8. The programs that were carried over from a past project are shown in black. From the plot, it is clear that the mean finish times for the two types of programs are different*. A t-test of the null hypothesis that the two mean finish times are equal is presented in Table 3-3. This analysis confirms that the two types of programs have significantly different average finish times at the $\alpha = .1$ level.

![Histogram of all program duration for Project 1. Carry-over programs are shown in black.](image)

* Figure 3-8

Also note that the carry over programs appear to be separated into two groups, as described earlier.
Table 3-3  Summary statistics for program finish time for programs in Project 1.

<table>
<thead>
<tr>
<th>Program Type</th>
<th>Mean</th>
<th>Variance</th>
<th>N</th>
<th>Result of t-test for ( \mu_{(Carry)} - \mu_{(New)} = 0 )††</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>2.2960</td>
<td>19.317</td>
<td>223</td>
<td>( t_{calc} = 6.81 ) with 290 df</td>
</tr>
<tr>
<td>Carry Over</td>
<td>-6.2609</td>
<td>80.725</td>
<td>69</td>
<td>Reject</td>
</tr>
</tbody>
</table>

For the test, \( t_{calc} = \frac{2.2960 - (-6.2609)}{\sqrt{19.317/223 + 80.725/69}} \). Similar analyses were performed for all other projects on which data was collected. The results of those analyses were also similar and confirmed the above results.

3.2.2 Linear Regression Analysis

Since it has been determined that the new and carry-over programs have significantly different average finish times, the programs will be partitioned according to their classification as new or carried over. Analysis of the historical data indicates that this partitioning provides increased predictive accuracy from a regression standpoint. A demonstration of how this partitioning is justified follows. Recall the scatter-plot presented in Section 1.3. The plot shows a strong indication of one “new” regression function, and possibly up to two “carry-over” regression functions.

†† The alternate hypothesis was \( \mu_{(New)} > \mu_{(Carry)} \) and the significance level was \( \alpha = .1 \).
Two models were fit to the data, and an $F$ test of the residual sums of squares was performed. The first model allows for two separate slope and intercept terms (for each type of program); we call this model the \textit{full} model. The second model contains only one slope and two intercept terms; we call this model the \textit{reduced} model. Both models are combinations of the basic theoretical linear model discussed in Section 2.3.4.

Formally, the full model is

$$y_i = \beta_{0n} + \beta_{0c} + \beta_{1n}x_{in} + \beta_{1c}x_{ic} + \varepsilon_{in} + \varepsilon_{ic}$$  \hspace{1cm} (3.1)$$

where $\beta_{0n}$ and $\beta_{0c}$ are the intercepts and $\beta_{1n}$ and $\beta_{1c}$ are the slopes. The "n" and "c" subscripts indicate that the observation comes from the "new" class or the "carry-over" class, respectively.

The reduced model is

$$y_i = \beta_{0n} + \beta_{0c} + \beta_{1n}x_i + \varepsilon_i$$  \hspace{1cm} (3.2)$$
where \( \beta_{0n} \) and \( \beta_{0c} \) are the intercepts as before, and \( \beta_1 \) is the common slope.

Using the two models, we can test the null hypothesis of the "homogeneity of slopes":

\[
H_0: \beta_{1n} = \beta_{1c} \tag{3.3}
\]

This hypothesis can be tested using the residual sums of squares, \( SS(\text{Res}) \), from the two models. For any general hypothesis the sum of squares for the hypothesis can be computed as

\[
Q = SS(\text{Res}_{\text{reduced}}) - SS(\text{Res}_{\text{full}}) \tag{3.4}
\]

where "reduced" and "full" identify the two models [Rawlings, 1988]. \( Q \) has degrees of freedom equal to the number of linearly independent constraints, \( k \), imposed by the hypothesis on the full model. In our case, this number is \( k=1 \). The computed F-statistic for the test is, in general,

\[
F_{\text{calc}} = \frac{Q}{\hat{\sigma}^2} \tag{3.5}
\]

where \( \hat{\sigma}^2 \) is the mean square error from the full model (equation 3.1).

Fitting the full model to the Project 1 data gave a residual sum of squares of \( SS(\text{Res}_{\text{full}}) = 9404.929 \); fitting the reduced model gave a residual sum of squares of \( SS(\text{Res}_{\text{reduced}}) = 9579.643 \). The difference is \( Q = SS(\text{Res}_{\text{reduced}}) - SS(\text{Res}_{\text{full}}) = 174.714 \) with 1 degree of freedom. The full model fit yields a mean square error of \( \hat{\sigma}^2 = 32.626 \) with 288 degrees of freedom. The calculated F-statistic is thus

\[
F_{\text{calc}} = \frac{Q}{\hat{\sigma}^2} = \frac{174.714}{32.626} = 5.355 \tag{3.6}
\]

with 1 numerator and 288 denominator degrees of freedom. This \( F_{\text{calc}} \) value is significant beyond the .05 level, so we reject \( H_0 \) and conclude that the two intercepts are different. Since it is known that the mean finish times for the two kinds of programs are significantly
different, one would expect the intercepts, $\beta_0n$ and $\beta_0c$, to be different for the models. In the presence of different slopes, the value of a test of homogeneity is questionable, unless we expect the regression lines to intercept at a certain point. This test will not be made with these data, since it is not clear that the two models should intercept at a common point. This is true for this data partitioning scheme, since the carry-over class contains programs that finished in another project, and so were aiming at another milestone date. This is addressed further in the next section.

**ANALYSIS OF THE PARAMETER ESTIMATES**

The single-variable model parameters are estimated in this section, and a discussion of how they conform to the assumptions and implications of the single variable model follows.

The General Linear Models (GLM) procedure available in the STAT™ subsystem of the Statistical Analysis System (SAS®)† was used to fit the model to the data [SAS 1990a], [SAS 1990b], [SAS 1988]. Table 3-4 shows the results of fitting the full model given in equation 3.1. Since the classification of programs by type partitions the data set, one could do separate analyses on each partition to estimate the parameters. The mean square error for this model would be equal to the pooled variance from all the separate models; this estimate is the best estimate of $\sigma^2$, unless a pure error estimate is available [Rawlings, 1988].

Figure 3-9.2 shows the raw data plotted with the fitted regression lines. The model fit is equation 3.1, with one line for the new programs and one for the carry-over programs.

---

† SAS, GLM and SAS/STAT are registered trademarks or trademarks, as indicated by their marks, of SAS Institute, Inc.
NEW PROGRAMS

From Table 3-4 we see that the estimate of the intercept for the new programs is 5.7613, and its 90% confidence interval overlaps neither the sample mean nor 0. The estimated slope for the new programs is −0.7960. Again, the expected value of −1 does not fall within the bounds of the 90% confidence interval. Obviously, the mean of the finish times for new programs is significantly greater than 0, which is an indication that the entire project is late. Equation 2.8 implies that the mean finish time will be greater than 0 if (1−a)E[X] + d > 0. This implies that it must be the case that a > d/E[x] +1, assuming that
E[X] < 0. We can estimate the quantity on the right by using the estimate for d (intercept) and the sample mean of the (new program) start times, E[X] = -16.996. This yields a value of 0.661; hence, a must be greater than 0.661, or -a < -0.661, which is the case, for these estimates, since -a is estimated by \( \beta_1 \) for the new programs.

**CARRY-OVER PROGRAMS**

Table 3-3 shows that the mean of the new program finish times is -6.2609. The estimate, -5.1383, for the carry-over programs intercept, \( \beta_{0c} \), falls close to the sample mean of the carry-over program finish times, -6.2609, and is, in fact, in the 90% confidence interval for the estimate; but it is significantly different from 0. Similarly, the estimate of -0.9701 for the carry-over programs slope, \( \beta_{1c} \), is not significantly different from -1 at the 90% level. It is necessary to note, however, that the carry-over program residuals behavior is questionable. The plots indicate that perhaps another variable important specifically to the carry-over programs is missing from the model. This is discussed in the next section.

Statistical tests and confidence interval statements using regression models are valid whenever the assumptions of linear regression are adequately satisfied by the data being analyzed. This question is addressed in the next section.

**TESTING THE REGRESSION ASSUMPTIONS**

In order to fully apply the single-variable regression model, it must also be established that the historical data adequately conform to the least squares assumptions. The assumption that is crucial to confidence intervals and hypothesis tests requires that the error term in the regression model has a normal distribution with mean 0. Also, the assumption that the model is correct was made. This means that no important independent variables have been omitted.(including any relationships that are nonlinear in the parameters). The data were analyzed to determine if the assumptions are adequately satisfied. Residuals behavior and
other diagnostic techniques were used. Only the analyses for the Project 1 data are given. Results from four other projects gave similar results.

Figure 3-10 shows a histogram of residuals for the full model fit (equation 3.1). The total mass of residuals appears to be distributed approximately normally. However, the residuals associated with the carry-over programs (shown in black) are non-normal, and have a bi-modal distribution. This might indicate that other factors may be important for the carry-over programs, such as a schedule slip, or the fact that some of them finished in a previous project. A 2-time-unit slip was announced at time 0 in Project 1. Taking either effect into account might improve residual behavior. The factor may also improve the precision of the estimates of the model parameters.

![Histogram of the residuals from the full model regression for Project 1. The black areas indicate carry-overs.](image)

**Figure 3-10** Histogram of the residuals from the full model regression for Project 1. The black areas indicate carry-overs.

A normal probability plot of the residuals is shown in Figure 3-11. It indicates a near-normal distribution for the entire residual set. The bumps in the curve indicate that there are deviations from the expected shape. Coupled with the observed poor carry-over residual behavior, this probably indicates an inadequate model.
The property that the carry-over programs lie on the periphery of the residual plots can also be seen in the scatterplot of the residuals versus the predicted values. This is shown in Figure 3-12. The plot indicates a good residual behavior, in the sense that there is no discernable dependence on the mean of the dependent variable (e.g., fanning out). This also supports the assumption of equal variance of the error terms.

Figure 3-11 Normal probability plot of the residuals from the full model regression for Project 1.

Figure 3-12 Scatterplot of the residuals vs. the predicted values from the full model regression for Project 1. The x's denote the carry-overs.

Figure 3-13 shows the Cook's $D_i$ statistic plotted against the observation numbers of the data. Cook's $D_i$ is a measure of the potential influence of a given point, i. Cook's $D_i$
measures the effect that deleting the \( i \)th observation will have on all the parameter estimates. Again, the carry-over programs appear problematic since the greatest number of significant Cook's \( D_i \) values are due to carry-overs. One recommended cutoff value for the statistic is \( 4/n \), where \( n \) is the number of observations used. In this case, the cutoff value is \( 4/292 = 0.014 \). A relatively high number of the carry-over points are significant.

![Figure 3-13](image)

**Figure 3-13** Scatterplot of the Cook's \( D_i \) vs. observation number from the full model regression for Project 1. The x's denote the carry-overs.

### 3.2.3 Another Data Partitioning Scheme

The previous model suffers from poor residuals behavior, and, according to the Cook's \( D_i \) statistic, has many potentially influential points. The poor behavior is mostly attributable to the carry-over programs. In an attempt to improve the behavior of residuals (and therefore improve the adequacy of the linear regression model), the programs were partitioned into new and carry-over in the same way as before, but the carry-over programs were also partitioned. The partitioning depended on when the carry-over programs finished. If a program finished before the actual finish date for the previous project, then it was placed in partition a); otherwise, it was placed in partition b) (see Section 3.2.1).
Justification for this division comes mostly from circumstantial evidence obtained from company employees. However, the division does not seem to be artificial, if one studies the plots with the data in hand. It could be that the carry-overs that finished in the previous project are actually developed in the previous project, and tested and/or delivered in the current project. It may also be the case that those programs were developed as a platform on which other programs can be written. This would partially invalidate the assumption that the programs are developed and proceed independently. The extent to which these conditions hold is still unclear, and warrants further investigation.

All the start times for the regression were measured from the planned finish milestone date of the current project. It could be argued that for the carry-over programs that were finished in the previous project, the start times should be measured with respect to that project's planned finish milestone date. However, the only estimate that changes depending on which milestone is used is the intercept term. *This is only true after the fact.* That is, when one wants to make a future prediction, then the program start times will have to be measured with respect to their own project's finish milestone date, since that milestone affects their behavior. Thus, to make predictions requires accurate information on which projects the programs are intended to be developed, and which projects they are intended to be released.

A model similar to the full model in equation 3.1 was used:

$$y_i = \beta_0n + \beta_0c_a + \beta_0c_b + \beta_1nX_{in} + \beta_1c_aX_{ia} + \beta_1c_bX_{cb} + \epsilon_{ij}$$ (3.11)

As before, "n" represents the new programs. Now, \(c_a\) and \(c_b\) identify the two kinds of carry-over programs described above. For this model, \(c_a\) represents a carry-over that was finished in the previous project (type a), and \(c_b\) represents a carry-over that was finished in this project (type b). Figure 3-14 shows the design duration plotted as a function of the
start time. The fitted regression lines from equation 3.11 are also indicated. The data points are identified with respect to the classification described above.

![Legend](image)

**Figure 3-14** Results of fitting equation 3.11.

A summary of the regression using the model defined in equation 3.11 is shown in Table 3-5.

**Table 3-5** Regression summary for program duration vs. start time for programs in Project 1, with the new partitioning of carry-over programs.

<table>
<thead>
<tr>
<th>Source</th>
<th>df</th>
<th>Sum of Squares</th>
<th>Mean Square</th>
<th>$F_{calc}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>6</td>
<td>177188.354</td>
<td>29531.392</td>
<td>1737.62</td>
</tr>
<tr>
<td>Error</td>
<td>286</td>
<td>4860.646</td>
<td>16.995</td>
<td></td>
</tr>
<tr>
<td>Uncorr. Total</td>
<td>292</td>
<td>182049.000</td>
<td></td>
<td>$r^2 = 0.8754$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
<th>Std. Error</th>
<th>Lower</th>
<th>Upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta_{0c_a}$</td>
<td>-15.1090</td>
<td>2.5334</td>
<td>-10.9416</td>
<td>-19.2764</td>
</tr>
<tr>
<td>$\beta_{0c_b}$</td>
<td>2.6932</td>
<td>1.4953</td>
<td>0.2334</td>
<td>5.1530</td>
</tr>
<tr>
<td>$\beta_{0n}$</td>
<td>5.7613</td>
<td>0.8048</td>
<td>4.4374</td>
<td>7.0852</td>
</tr>
<tr>
<td>$\beta_{1c_a}$</td>
<td>-1.0160</td>
<td>0.0640</td>
<td>-1.1213</td>
<td>-0.9107</td>
</tr>
<tr>
<td>$\beta_{1c_b}$</td>
<td>-0.9744</td>
<td>0.0356</td>
<td>-1.0330</td>
<td>-0.9158</td>
</tr>
<tr>
<td>$\beta_{1n}$</td>
<td>-0.7961</td>
<td>0.0445</td>
<td>-0.8693</td>
<td>-0.7229</td>
</tr>
</tbody>
</table>

The next set of diagnostic plots show improved residual behavior, especially with respect to the carry-over programs. The residuals still appear a little skewed towards the negative,
though. This could be due to the fact that a schedule slip of 2 units occurred at time 0, and it is not being accounted for. This effect would further separate the new programs from the carry-over programs, since the carry-over programs tend to finish earlier on average. The plot of the residuals versus predicted values, and the plot of residuals by the groups, provides further evidence that the variance of the error terms is equal across all levels of the factors.

![Histogram of residuals from the model in 3.11 for Project 1 with the new division of carry-over programs.](image)

**Figure 3-15** Histogram of residuals from the model in 3.11 for Project 1 with the new division of carry-over programs.
**Figure 3-16** Normal probability plot of the residuals from the model in 3.11 for Project 1 with the new division of carry-over programs.

**Figure 3-17** Scatterplot of the residuals vs. predicted values from the model in 3.11 for Project 1 with the new division of carry-over programs.
Figure 3-18 Plot of the residuals by group from the model in 3.11 for Project 1 with the new division of carry-over programs.

Figure 3-19 Plot of the Cook’s $D_i$ statistic by group from the model in 3.11 for Project 1 with the new division of carry-over programs.

The Cook’s $D_i$ statistic shows that few of the points are still influential. Further investigation yielded no reason for discarding the points, partly because similar points are present in all of the investigated data sets. It could be the case that all such points are in error, however, but further investigation would be required before ruling the points out altogether. The points, however, require special attention when performing hypothesis tests.

Of interest is the slope of the regression lines for the programs that were to be finished in this project. This includes both new programs and carry-over programs. If the slopes are
different, then this might be a good indicator of a degree of lateness for either class of program. The hypothesis is

$$H_0: \beta_{1n} = \beta_{1c_2} \quad (3.12)$$

By substituting a common slope term for the two shown above in equation 3.11, we obtain the reduced model. When fit to the data, it gave $SS(Res) = 5027.063$, with 287 degrees of freedom. From the full model, $SS(Res) = 4860.646$, with 286 degrees of freedom, for a mean square error of 16.995, which is the denominator in the calculated $F$ statistic. The calculated $F$ statistic for the test is

$$F_{calc} = \frac{(5027.063 - 4860.646)/1}{16.995} = 9.79$$

with 1 numerator and 286 denominator degrees of freedom. This is significant beyond the .05 level, so we conclude that the two slopes are different.

To investigate the potentially influential points further, the two models were fit to the data, omitting the three most potentially influential points (those with $D_i > 0.2$). The test gave an $F_{calc} = \frac{(4771.436 - 4364.322)/1}{15.422} = 26.40$, with 1 numerator and 283 denominator degrees of freedom. This $F$ value is significant beyond the .05 level, so the conclusion is the same: the two slopes are different.

**DISCUSSION OF THE TWO MODELS**

The first full model, defined by equation 3.1, that was fit to the data used class variables that were directly obtained from a database. The second model, defined in equation 3.11, however, used class variables that were based on the division of the carry-over programs. The database used to obtain data did not have an explicit field that definitively classified carry-over programs. The latter model's approach places a linear constraint on the finish times of carry-over programs. Motivation for the latter approach stems from discussions with company employees, and it deserves adequate attention in the database, as well. This claim is based on the fact that the second model fits the data better. The fit is better in the sense that it showed improved residual behavior, and the error sum of squares was
reduced. The latter fact would mean smaller error bounds on predictions made with the model.

The second full model is also more consistent with the basic ideas of the linear model in this study. It separates out the programs according to the schedule they were developed under. For predictions, this information is vital. The disadvantage of the (current) second model is that the information used to separate the carry-over programs may not be known early in the current project. Maintaining accurate database information would be vital to the practical use of the second model.
4 CONCLUSION

The next two sections summarize the accomplishments of this work and give possible topics for future research.

4.1 Summary

The main goal of this study was to provide a description, in the form of a regression model, of the behavior of large software projects developed under a milestone date driven process. Other investigations and accomplishments included

- Possible implications of Parkinson’s Law, the Deadline Effect and self-fulfilling prophecies were quantified. Essentially, these effects indicate that work will consume the allotted time. If the above effects play a major role in a project, then the model discussed in this report can be used to describe the average behavior of the project.

- Assuming the product component life-cycle phase durations behave in a stochastic manner, the durations are related to the time available to a planned milestone date via a regression function. This was used to describe product component completion behavior for software project phases.

- A linear regression model was used to illustrate the model and to demonstrate its adequacy in an observed environment.

- The model was evaluated using both empirical project data and simulations. Simulations also served as a means of investigating effects that cannot be easily derived analytically, such as schedule changes.

- Model expansion and programmability have been facilitated, helping to ensure that the model paradigms will be investigated further, and can be adapted and instituted in different environments.

- Examples of how to incorporate more product-related factors into the model in the form of class variables were shown. The choice of model-driving parameters
was limited to those that could be known early in a project, to facilitate making predictions early in a project.

The linear model provided an illustration of the more general regression approach. The linear model, using least squares to estimate the model parameters, has limitations. The limitations include

- The general regression model (Section 2.2.3) implies that program phase durations are non-negative. The linear model may produce negative duration predictions near the intercept on the predictor variable axis, if the normality of errors is assumed. The normality assumption is not required to estimate model parameters by least squares, but it provides the basis for testing hypotheses and forming confidence intervals. Using a segmented model, a non-linear model, or another estimation technique (like maximum likelihood) may eliminate this basic inadequacy. Transforming (e.g., log) the data to ensure only positive durations is another possibility.

- The model assumes that the project schedule is "realistic". This means that Parkinson's Law and the Deadline Effect can play a significant role in meeting deadlines on time. Schedule changes occur often in real projects. This topic was addressed in Section 2.4 and Section 3.1.2.

- The model is memoryless. It does not presently incorporate information on the past performance of components with respect to previous milestones. If a component did not achieve a previous milestone on time, it may increase its chance of not achieving future milestones on time. This is discussed in the next section.

This research resulted in the following conclusions.

- Specific software process and project schedule information cannot be separated when describing software project behavior.
• Accurate, company-specific data collection must be performed. The data to be collected must include resource (effort) and schedule-related information, including detailed accounts of schedule changes (e.g., when deadlines are changed, what they are changed to and from).

• Companies in the software development business should devote time and resources specifically to software process modeling. Company-specific findings should be used to tailor work already done in the area by comparing and contrasting them.

• The model given in this report attempts to combine process, schedule and product factors in order to describe the way product components finish life-cycle phases. This could aid software managers in analyzing risks associated with not meeting planned project deadlines.

It is a sincere hope that this work emphasizes further the need for process modeling in the software industry. In order to deliver quality software in the shortest time possible, one needs a good understanding of the specific process under which it is developed. Deriving models of the process can help this effort.

4.2 Recommendations for Future Research

The model presented in this report needs to be investigated further. It is a macro model at the product level. More information (e.g., from COCOMO) about the product’s components could be incorporated to reduce variability in the model. The model could be extended in this way to “focus in” on both the time aspect and the product aspect.

This work lays natural groundwork for the development and implementation of a predictive model. Although the design phase was emphasized as a whole, it is possible to model each subphase in a similar way. Information gained at successive milestones could be incorporated into a predictive model to supplement predictions.
It was also found to be enlightening, in this environment, to plot, for each program in the product, the dates they achieved a given milestone against the date they achieved the previous milestone. Similar plots can be made for not only the previous milestone, but also for the one before that, and so on. The observed correlation between the two quantities tended to decrease as the milestones used became further apart in the sequence. Examples of such plots from this environment follow below in Figure 4-1.

![Figure 4-1 Example plots of times programs achieved various milestones versus the time they achieved the first milestone.](image)

Measuring the dates with respect to the planned milestone date for their respective phases might help predict which components would be late, based on their achievement of previous milestones. As before, the correlation decreases in these analyses when measured over more than one interval, but other relationships may exist. Performing paired comparisons with programs at various milestones may prove useful in determining
probability relationships. That is, it may provide information that could lead to making statements like “A component has an 80% chance of being late for milestone Y, since it was late at milestone X”, where milestone X precedes milestone Y.

The variables that were investigated were specific to one company’s environment. It would also be useful to investigate the utility of other qualitative variables, both for this specific environment, and for others. The new/carry-over division presented in Section 3.2 helped to reduce error variability. Other quantitative variables, like lines of code, could be investigated, and included in the model, tailored to specific environments.

Further investigation of the effects of schedule changes should be made. Change means both schedule contraction or relaxation (slippage). Accurate data must be collected. Some possible explanations for schedule slips were given in Section 2.4. The explanations are consistent with the linear model approach, and were investigated in Section 3.1.2. The effect of the change on the developers that have already begun working, but have not finished by the change announcement, should be given special attention.

Finally, independent validation of the model assumptions should be performed. The results would help to better understand the applicability of the model. It could be that the model is valid only for the specific environment it was developed to describe.
5 REFERENCES


APPENDIX I: Confidence Intervals For LSQ Fit

Suppose we have used the method of least squares to fit a linear regression curve of the form

\[ y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \]  \hspace{1cm} (A1.1)

to a set of \( N \) data points to obtain estimates of the slope and intercept terms: \( \hat{\beta}_0 \) and \( \hat{\beta}_1 \), respectively. Assuming the random errors, \( \varepsilon_i \), are normally distributed with equal variance, we can deduce confidence intervals for these two parameters, as well as for the estimated mean of \( y \), given \( x \). We can also calculate confidence intervals for predicted values of \( y \), \( y_{pred} \). The formulas are summarized below [see Musa et al, 1987].

A 100(1-\( \alpha \)) percent confidence interval for a statistic, \( \theta \), is expressed below as

\[ \hat{\theta} \pm t_{\alpha/2, v} \text{s.e.}(\hat{\theta}) \]

where \( \hat{\theta} \) is an estimate of \( \theta \), \( \text{s.e.}(\hat{\theta}) \) is an estimate of the standard error of \( \theta \), and \( t_{\alpha/2, v} \) is the \( \alpha/2 \) percentile of the t distribution with \( v \) degrees of freedom. Let \( \hat{\sigma}^2 \) be an unbiased estimate of the variance, \( \sigma^2 \), of the error terms. An estimate can be obtained by:

\[ \hat{\sigma}^2 = \frac{1}{N} \sum_{i=1}^{N} \frac{(y_i - \bar{y})^2}{N-2} \]  \hspace{1cm} (A1.2)

- A 100(1-\( \alpha \)) percent confidence interval for the intercept term, \( \beta_0 \):

\[ \hat{\beta}_0 \pm t_{\alpha/2, N-2} \hat{\sigma} \sqrt{\frac{\sum_{i=1}^{N} x_i^2}{N \sum_{i=1}^{N} (x_i - \bar{x})^2}} \]  \hspace{1cm} (A1.3)
• A 100(1-\(\alpha\)) percent confidence interval for the slope term, \(\hat{\beta}_1\):

\[
\hat{\beta}_1 \pm \frac{t_{\alpha/2, \, N-2} \hat{\sigma}}{\sqrt{\sum_{i=1}^{N} (x_i - \bar{x})^2}}
\]  
(A 1.4)

• A 100(1-\(\alpha\)) percent confidence interval for the mean of \(y\), given a value \(x\):

\[
\hat{y}_0 \pm t_{\alpha/2, \, N-2} \hat{\sigma} \sqrt{\frac{1}{N} + \frac{(x_0 - \bar{x})^2}{N \sum_{i=1}^{N} (x_i - \bar{x})^2}}
\]  
(A 1.5)

• A 100(1-\(\alpha\)) percent confidence interval for a predictions of a dependent variable, \(y_{\text{pred}}\), for a value of \(x_0\):

\[
\hat{y}_{\text{pred}} \pm t_{\alpha/2, \, N-2} \hat{\sigma} \sqrt{1 + \frac{1}{N} + \frac{(x_0 - \bar{x})^2}{N \sum_{i=1}^{N} (x_i - \bar{x})^2}}
\]  
(A 1.6)