

/SOFTWARE DEVELOPMENT RESOURCE ESTIMATION
IN THE 4TH GENERATION ENVIRONMENT/

by

DANIEL RAYMOND MOYER

B. S., Bowling Green State University, 1971

A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

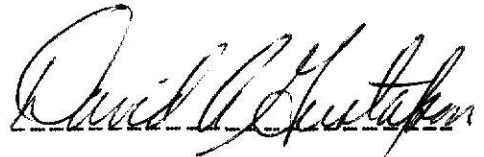
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

Approved by:



Major Professor

LD
2668
.R4
1986
m63
c.2

TABLE OF CONTENTS

A11202 663944

CHAPTER 1 RESOURCE PLANNING IN SOFTWARE DEVELOPMENT

| | |
|---|---|
| 1.1 Overview | 1 |
| 1.2 Background and Purpose | 1 |
| 1.3 Historical Overview | 2 |
| 1.4 Current and Future Requirements | 5 |

CHAPTER 2 THE PUTNAM MODEL

| | |
|------------------------------------|----|
| 2.1 Overview | 7 |
| 2.2 Rayleigh/Norden Equation | 8 |
| 2.3 The Software Equation | 12 |

CHAPTER 3 RESOURCE ESTIMATION USING 4TH GENERATION LANGUAGES - A NEW ERA

| | |
|---|----|
| 3.1 Overview | 18 |
| 3.2 The Putnam Model in the 4th Generation Environment | 19 |

CHAPTER 4 GRAPHIC PORTRAYAL OF THE IMPACT OF 4TH GENERATION TECHNOLOGY OF THE PUTNAM MODEL

| | |
|--|----|
| 4.1 Background | 27 |
| 4.2 Hierarchical Representation of Program Modules ... | 28 |
| 4.3 Module Specifications for Program Curveplot | 29 |
| 4.4 Pseudo-code for Program Curveplot | 34 |

CHAPTER 5 A SUMMARY OF CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

| | |
|-----------------------|----|
| 5.1 Overview | 38 |
| 5.2 Conclusions | 38 |

| | |
|--|-----|
| 5.3 Recommendations | 39 |
| APPENDIX A SOURCE CODE FOR PROGRAM CURVEPLOT | A-1 |
| BIBLIOGRAPHY | |

FIGURES AND TABLES

| | | |
|--------------|---|----|
| Table 1.3.1 | Comparitive Metric Values | 3 |
| Figure 2.2.1 | Rayleigh/Norden Manpower Utilization Curve .. | 9 |
| Figure 2.2.2 | Lifecycle with (a) constant | 11 |
| Figure 2.2.3 | Lifecycle with (K) constant | 11 |
| Figure 2.3.1 | Putnam's Difficulty Measure | 13 |
| Figure 2.3.2 | Putnam's Software Lifecycle | 16 |
| Figure 3.2.1 | Lifecycle Curve with (K) constant | 21 |
| Figure 4.2.1 | Program Curveplot Hierarchy Diagram | 28 |

ACKNOWLEDGEMENTS

The author would like to take this opportunity to gratefully express his sincere appreciation to the following people for their assistance, support, and respective roles in the successful completion of this project. To my Graduate Committee: Dr. David Gustafson, Major Professor, for his support, encouragement, patience, guidance, and friendship. Dr. Paul Fisher, for his guidance, continuing support, understanding, and friendship over the years. Dr. Austin Melton, for his willing support and assistance in the completion of this undertaking. To CPT George Sherman, a loyal friend, whose encouragement, friendship, and editing assistance were invaluable. To Mr. John D. Wiggins, Sr., for his editing expertise and willingness to assist in ensuring a polished finished product.

This accomplishment is dedicated to my family and to Glenda Tullous, Michelle, and Jennifer. Their love, moral support, encouragement, and understanding were the foundation upon which this accomplishment was based and for which I will be eternally grateful.

CHAPTER 1 RESOURCE PLANNING IN SOFTWARE DEVELOPMENT

1.1 OVERVIEW.

"You cannot control what you cannot measure." [Dem 83]

With this succinct statement Tom DeMarco provides a strong justification for the study of project size estimation in the software development process. In the course of this report the author intends to provide a historical overview of the most widely used measure developed to fulfill this requirement, the Putnam Model, to provide the foundations upon which this measure is based, to detail a methodology for automating this software development tool, and to analyze the impact of 4th generation technological advances upon this tool.

1.2 BACKGROUND AND PURPOSE.

Software development management has historically been a fruitful area of study for the software engineer for a variety of reasons. Perhaps the most cogent rationale for the emphasis placed on the study of the software development management process and upon the science of software engineering is the fact that this area, perhaps more so than any other area of computer science, is the principal interface with the ultimate beneficiary of the discipline - the customer or end-user of the product. While customers are rapidly becoming more sophisticated and knowledgeable in this generation of computerization, the fact remains that in the final analysis the customer's principal interest is in the deliverable product and not in how or why the

product works. For this reason the software engineer's role becomes paramount in importance for it is he who begins the process of determining and interpreting requirements, specifying those requirements, designing the system to fulfill those requirements, and monitoring or participating in the coding, testing, and implementation of the finished software product. In a traditional business environment it is precisely the software engineer's role in this process which may well determine the success or failure of a software development project from its inception. Business, in general, has little patience with project personnel speaking principally in technical jargon or with inefficient project planning. The software development plan becomes a vital document in project planning. Project estimation is a crucial link in successful completion and acceptance of the deliverable product. The science of software engineering will therefore play a fundamental role in software development management for the foreseeable future.

1.3 HISTORICAL OVERVIEW.

With the importance of the software engineer's role in the software development management process comes a corresponding increase in responsibility for providing the most accurate estimates of project size and complexity as is feasible. Historically, the primary emphasis in the area of resource estimation has been one of volume in terms of time required and personnel required for project completion. Size estimation and project duration estimation at a given staffing level have been

**THIS BOOK
CONTAINS
NUMEROUS PAGES
WITH DIAGRAMS
THAT ARE CROOKED
COMPARED TO THE
REST OF THE
INFORMATION ON
THE PAGE.**

**THIS IS AS
RECEIVED FROM
CUSTOMER.**

the focus of this effort. For this reason, the traditional standard used in resource estimation has been that of Lines of Code (LOC). This is found in a review of the works of Putnam [Put 78] [Put 79], Bailey and Basili [Bas 81], Walston and Felix [Wal 77], and to a far lesser degree, DeMarco [DeM 82].

Putnam's works, and the formulas upon which they are derived are the topic of this report; as such they will be deferred to later analysis in this report. However, to give the reader a greater appreciation for the disparate range of values that the referenced metrics may produce, the table below provides the reader with an appreciation for the wide range of values the traditional metrics may provide based on fixed values of 5,000, 50,000, and 100,000 deliverable lines of source code for a given project.

| | 5000 | 50000 | 100000 |
|--|-------|--------|--------|
| Bailey & Basili 1.17 ($E = 3.4 + .72 * DL$) | 15319 | 226534 | 509724 |
| Boehm 1.05 ($MM = 2.4(KDSI)$) | 18371 | 206125 | 426787 |
| Walston & Felix .91 ($E = 5.2 L$) | 12080 | 98190 | 184503 |

Table 1.3.1 Comparative Metric Values

The principal reason for the omission of DeMarco in table 1.3.1 is the fact that DeMarco bases his metric upon the "Bang" measure and function points as opposed to deliverable lines of source code as a quantifiable measure.

Based principally upon empirical data gathered from completed projects, software developers have successfully correlated data on estimates for project size, project duration, and project costs. While this methodology is not in and of itself a bad practice, it is obvious that many pitfalls await the inexperienced practitioner in resource estimation due to the variety of factors inherent in utilizing these techniques. For example, the experience level of the project estimator most certainly plays a significant role in this process. One can readily imagine the disparity of LOC estimates which could be expected between a person estimating his first project as opposed to the estimate of an experienced resource estimator. Additionally, LOC estimates alone are not always a true indicator of total effort required for project completion. Projects involving embedded systems or software pose significantly more complex coding and design requirements than does a stand alone system of equivalent size. For these reasons, the historical development of software cost estimation measures has been based upon the incorporation of not only size but complexity, past experience with similar projects, and other factors as well to provide the software engineer with tools upon which to base his estimates. Of interest here is that the traditional measures

have been directed primarily at determining project size and project duration. Seldom are these estimation techniques associated with specific dollar costs in the literature. The reason for this is obvious. Factors such as inflation, the experience level of project personnel, and pay scales for development team personnel vary significantly among companies in the software development market. Therefore, specific dollar costs are extremely difficult to accurately project. Attempts to directly correlate projected dollar costs using generalized estimation formulas is a self-defeating proposition. Project size, complexity, and duration are far better resource indicators in the general approach to estimating project development requirements. For these reasons, software scientists have developed generalized formulas to produce LOC estimates, such as Putnam's software equation, man-month estimates, such as Boehm's COCOMO model, and indicators of complexity, such as Halstead or McCabe's metrics, in order to accurately estimate resources required for project completion.

1.4 CURRENT AND FUTURE REQUIREMENTS.

The area of resource estimation is of vital concern to the Software Engineer. Continued research and refinement of resource estimation techniques is essential to the Software Engineering discipline and to the business community. With the rapid growth of computer technology in virtually every facet of today's society, the role of software development companies is ever

expanding. In order to successfully meet the needs of corporate America as well as to remain commercially viable in this highly competitive field, it is imperative that these companies employ sound software engineering principles in their daily activities. Accurate, timely software development measures must be available for use by the software developers. Corporate America will fast lose patience with and confidence in any business where instances of 300% cost overruns and 200% scheduling overruns are the rule rather than the exception [Mar 83]. The importance of accurate, timely project estimation in the software development lifecycle cannot be overstated. The credibility of the software engineering discipline as well as the commercial viability of many software development companies may well revolve upon the ability to properly plan, develop, and estimate project size with a reasonable degree of accuracy and consistency. The advent of code generators and 4th generation languages has assisted in meeting the challenges of some of the aforementioned problems. However, they have also simultaneously generated new problems in terms of the technological impact of these tools upon traditional resource estimation formulas. Therefore, resource planning for software development is a fertile area for study and research. A review and analysis of the Putnam Model and associated resource estimation techniques is the topic of the following chapter.

CHAPTER 2 THE PUTNAM MODEL

2.1 OVERVIEW.

During the decade of the 70's, software engineers and software development managers searched for a model which adequately portrayed the behavior of the software development process in terms of resource expenditures. A great deal of research was devoted by a number of experts in this area. Many theoretical and empirical models were advanced during this period. One of the significant contributions to our understanding of the software development process was made by Lawrence Putnam during the mid to late 70's. During this period, Putnam produced a series of articles which demonstrated that software development projects tended to follow a predictable, well-defined lifecycle pattern. Putnam's observations of the Software Lifecycle were based upon the early works of Peter Norden [Nor 63] who demonstrated the mathematical formulas for deriving manpower utilization curves for research and development projects. After researching and collecting data from hundreds of medium to large scale software development projects, ($> 80,000$ lines of source code), Putnam found that the manpower utilization for these projects, in terms of project development time, coincided quite nicely with the Rayleigh/Norden utilization curves previously advanced. Further, Putnam discovered that by enhancing the developmental curve formulas with factors such as programmer productivity, source code statements required, and

technology constants, an empirical model for software development could be derived. Quite naturally, this model has been titled the Putnam Model. This model remains one of the most widely used and accepted models within the software development community. In the period of time since it was first published in 1976, much research has been done and many articles have been written advancing new hypotheses concerning the software lifecycle as described by Putnam. However, most are firmly based upon Putnam's original work and are extensions or enhancements of the original concepts. Given the importance of Putnam's contributions to our understanding of the software lifecycle, it is necessary to examine the fundamental components of the model. This will be the focus of the remainder of this chapter.

2.2 THE RAYLEIGH/NORDEN EQUATION.

As was previously stated, Peter Norden [Nor 63] advanced his manpower utilization theories in the early 60's based upon his observations of research and development projects performed at the I.B.M. Development Laboratory, Poughkeepsie, New York. Norden's research indicated that there was a distinct pattern of manpower utilization throughout the duration of these projects. His observations were that there are repeated cycles of manpower build-up, peaking, and phase-out corresponding to phases of complex projects. Further, Norden stated that these cycles could be represented as a mathematical function of the form:

$$Y' = 2 K a t e^{-at^2}$$

Where

y' = Manpower utilized each time period

K = total man-months of effort utilized by the end of the project

a = shape parameter (governing time to peak manpower)

t = elapsed time from start of cycle

Additionally, Norden indicated that, by repetition of application of the equation to coincide with the number of project cycles, a total project curve could be derived. This project curve was very similar to the distribution pattern of the classic Rayleigh Curve, thus the evolution of the Rayleigh/Norden Manpower Utilization Curve.

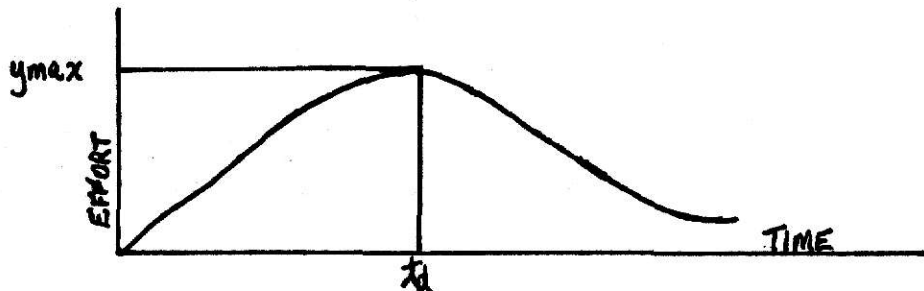


Figure 2.2.1 Rayleigh/Norden Manpower Utilization Curve

Putnam's analysis of empirical data collected on completed projects corresponded quite nicely with the Rayleigh/Norden form. However, in further analysis, Putnam determined that the shape parameter, (a), in the Rayleigh/Norden form exhibited a dependency upon the point in time at which (y') reached its peak. Putnam chose to represent point in time as (t_d). He mathematically derived that:

$$a = \frac{1}{2} (t_d)^2$$

Where (t_d) is the time to reach peak effort.

Putnam subsequently observed that (t_d) corresponded very closely to delivery time for a software development project. Substituting for (a) in the original lifecycle equation and using the following substitution sequence:

Putnam's derived substitution for (a)

$$a = 1/2 (t_d)^2$$

Equation for the entire lifecycle

$$\dot{Y} = K/t_d^2 \cdot t \cdot e^{-t^2/2t_d^2}$$

To find the area under the curve this formula is integrated in the form:

Definite Integral

$$Y = K(1 - e^{-t^2/2t_d^2})$$

Substituting the endpoints 0 and (t_d) to get the form of:

$$Y = K(1 - e^{(-t^2/2t_d^2)})$$

$$Y = K(1 - e^{-.5})$$

Therefore:

$$Y = .3935$$

Thus, DE is .4 of the total effort.

Since both parameters (K) and (a) may represent a range of values

for manpower and shape, it is obvious that varying these parameters permits the portrayal of a wide variety of shapes and magnitudes [Put 78]. Figure 2.2.2 demonstrates the effect on the lifecycle curve by maintaining (K) as a constant value of 1000 while Figure 2.2.3 demonstrates the effect of maintaining the value of the (t_d) constant and permitting the (K) value to vary.

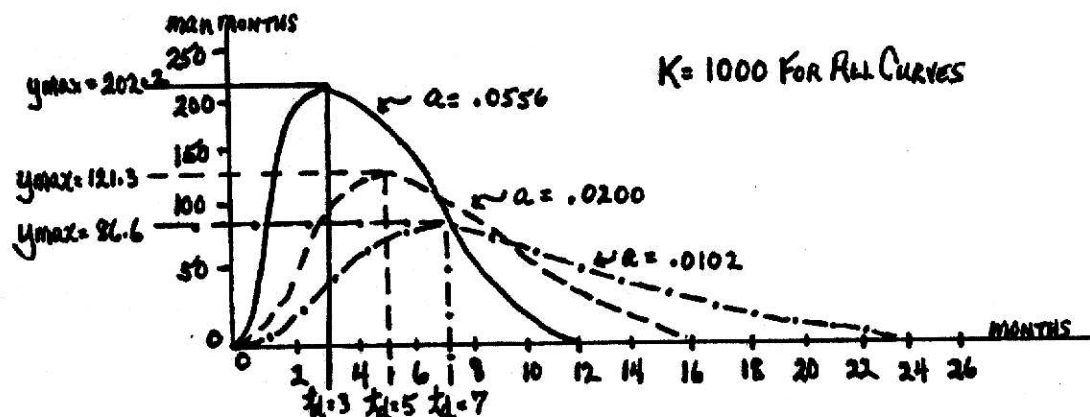


Figure 2.2.2 Lifecycle with (K) constant. [Nor 63]

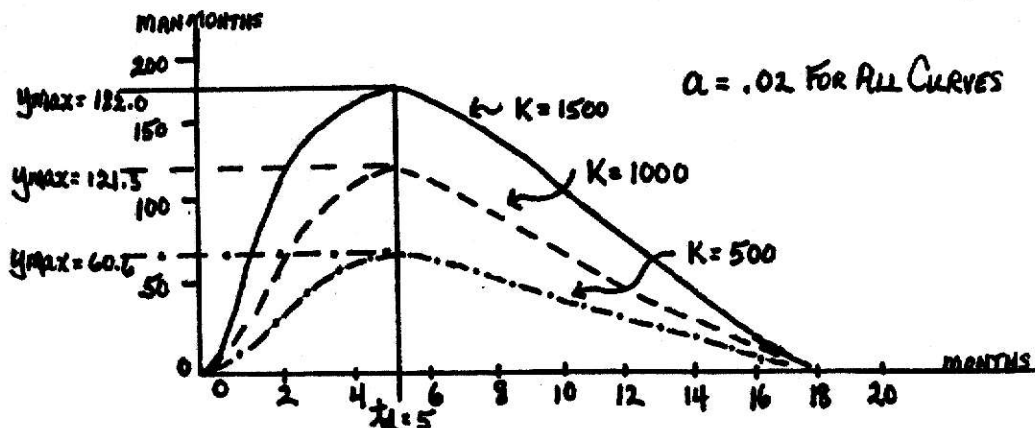


Figure 2.2.3 Lifecycle with (t_d) constant. [Nor 63]

Having analyzed and verified the data used to portray the development phase of the software lifecycle, Putnam began to analyze collected data to determine relationships from which to project effort required over the entire software lifecycle. The need for a quantifiable product, or formulas to correlate source

statements to LOC, was apparent. This was the conceptual basis for the development of Putnam's Software Equation.

2.3 THE SOFTWARE EQUATION.

Having demonstrated the effects of manipulating the management parameters, (K) and (t_d) , upon the lifecycle, Putnam directed his efforts to developing a relation which portrayed the relationship between these parameters and the deliverable product - source code. By means of "fitting" the data collected to the lifecycle curves(s), he was able to derive several relationships which would play significant roles in the ultimate formulation of the software equation. These relations are summarized in the following series of figures and discussion. First, Putnam rearranged the Rayleigh/Norden Curve via linearization in the form:

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

Putnam determined that when the natural logarithm of manpower divided by elapsed time is plotted against elapsed time squared, a straight line was formed with $(\ln(K/t_d^2))$ as the intercept and that $(-1/2t_d^2)$ was the slope for the line [Put 78]. He further discovered that if the number (K/t_d^2) was high in scale it corresponded to a difficult system implementation and that the reverse, if it were small in scale representing an easy system implementation, was also true. Thus the expression (K/t_d^2) represented system difficulty. This is graphically portrayed as:

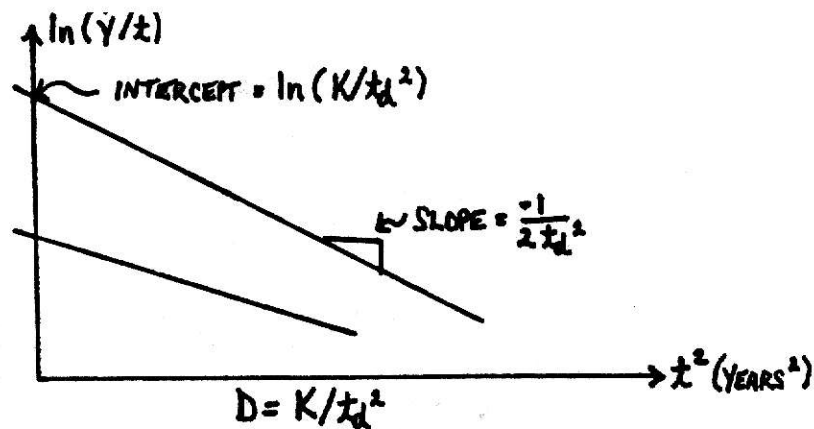


Figure 2.3.1 Putnam's Difficulty Measure

This difficulty measure has assumed a special significance over the years for it is used by Putnam to establish a fundamental relationship - that as development time is decreased, a dramatic increase in difficulty is to be expected. This relationship will be further analyzed in this report.

Putnam then turned his attention to formulating a productivity measure which would be essential to the determination of software code production rates. Based upon the data previously analyzed on completed software development projects, Putnam derived the following productivity relation:

$$\overline{PR} = C_n D^{-2/3}$$

Where

C_n = a productivity constant
 D = difficulty

It should be noted that (C_n) is a quantized constant representing a group of state-of-the-art technology constants, not a single productivity numeric value. Further, (\overline{PR}) is not a constant value, it varies and represents a range of values.

Having completed the linkage between productivity and

difficulty for code production, only one step remained to be determined prior to Putnam's formulation of the software equation. Putnam found that in relating effort required for code production a requirement existed for a weighting factor, or "burdened number", to account for the managerial overhead and test and evaluation portion of the coding cycle. Based upon his analysis of the "fitted data", he determined that the burdened number or scaling factor was to be 2.49 (\overline{PR}). Now all elements relating system size to productivity to effort were present. Through a process of substitution using the previously developed formulas and relations, demonstrated below, the software equation could then be derived:

$$S_s = 2.49 \cdot \overline{PR} \cdot K/6$$

$$= \frac{2.49}{6} C_n K^{1/3} t_d^{4/3}$$

Where

(S_s) = Delivered Lines of Source Code
 2.49 = Scaling Factor

Therefore, the Generalized Form of the Equation is:

$$S_s = C_K K^{1/3} t_d^{4/3}$$

Where

(C_K) subsumes $2.49/6$ (C_n) - A measure of
 human/machine/state-of-the-art technology system constant

Putnam's Software Equation

The final analysis performed by Putnam, of immediate, interest to this report, was the determination of a relation to

portray the relationship between effort expended and time expended during project development. This is of great importance to management in any business scenario and of particular import to software developers. This relation known as the effort - time tradeoff law can be deduced directly from the software equation in the following manner:

Given the software equation

$$S_s = C_K K^{1/3} t_d^{4/3}$$

Then

Constant source statements (S_s) implies ($K t_d^4$) Constant

So

$$K = \text{Constant} (t_d^4)$$

Or, recalling that Development Effort = (.4K)

Then

$$\text{Development Effort} = \text{Constant} / (t_d^4)$$

Thus Putnam has observed that development time for a system is compressible only down to some gradient difficulty condition. That is to say that adding manpower to a project in an effort to compress development time is feasible until a gradient difficulty condition is reached. The specific constraint determined to portray this phenomenon was:

$$K = | \nabla D | t_d^3$$

This implies that adding manpower to accelerate a developmental

project may be accomplished only at a very high cost and is attainable only to the point where the gradient condition is met. Thus, Putnam's findings corroborate Brook's Law, [Bro 75], which, simply stated, indicates that there is a point of diminishing returns when attempting to allocate manpower in an effort to reduce time expenditures for project completion.

It is conceded that this overview of Putnam's works is by no means exhaustive. However, the fundamental precepts upon which Putnam's Software Lifecycle are based have been adequately described. A pictorial representation of Putnam's Software Lifecycle follows:

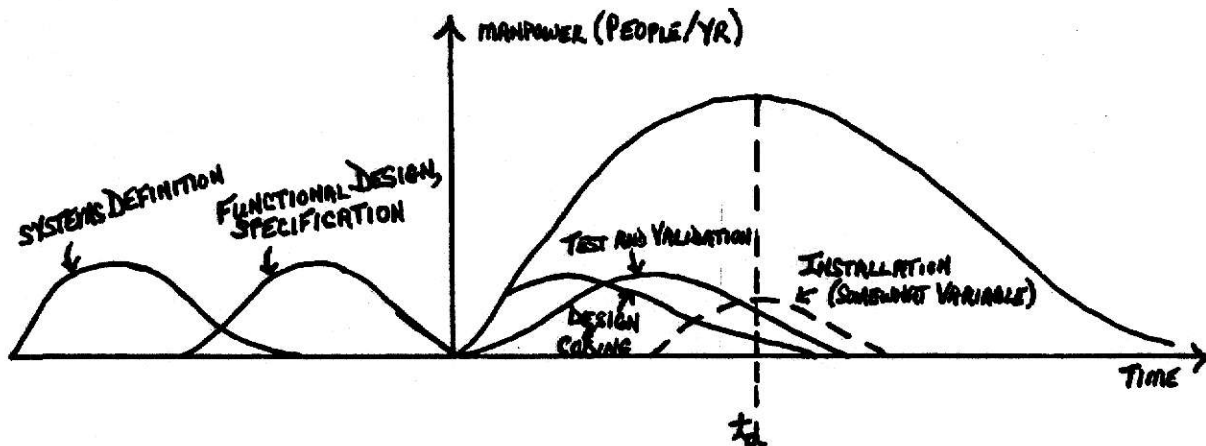


Figure 2.3.2 Putnam's Software Lifecycle

This concludes the analysis and summarization of the Putnam Model. The model has been empirically validated and shown to be quite accurate within the context of the technological environment in which it was developed and has been used. However, the advent of code generators and 4th generation languages has changed the traditional Putnam Model environment.

These changes mandate a reevaluation of the model in the context of the current environment. This will be the topic of the next chapter of this report.

CHAPTER 3 RESOURCE ESTIMATION USING 4TH GENERATION LANGUAGES

- A NEW ERA.

3.1 OVERVIEW.

Having reviewed in some detail the Software Lifecycle as originally developed in the traditional 3rd generation environment, the next logical step in the analytical process is to review the impact of subsequent technological advances upon the lifecycle. The author's interest in this area was generated through involvement in planning for a retail sales management project while working with Computer and Information Sciences, Inc., (CIS), a software development company based in Manhattan, Kansas. CIS had undertaken the retail sales management project for a ladies clothing distributor based in Dallas, Texas. Initial estimates of project size in terms of delivered lines of source code were in the range of 175,000 - 250,000 lines of code. Given the desired initial operational capability, (IOC), timeframe of approximately eight months, it was immediately apparent that a code generator or 4th generation language system would be required to ensure successful completion of the project. This precipitated an immediate search for a suitable tool to assist in the developmental process. Ultimately, two candidate systems were considered: 1) LINC, a Burroughs Corporation proprietary 4th generation language system and 2) PROGRESS, a Data Language Corporation proprietary 4th generation language system. Both systems produce ANSI Standard COBOL application

programs as the deliverable code product. Both systems claim increased productivity rates on the order of 10:1 to 20:1 and higher; that is to say, that 1 line of 4th generation source code will generate 20 lines of deliverable code in the application language. In a study performed at the University of Auckland, New Zealand, productivity rates of 40:1 and higher were documented on large scale projects using LINC [Rud 84]. Due in large part to its portability to a variety of hardware environments as well as product cost, PROGRESS was the tool selected for the retail sales management project. Unfortunately, the author's reassignment precluded data collection and analysis of the performance of the tool for incorporation in this report. However, taken in the context of potential resource savings to a small to medium sized software development company, these productivity increases are of enormous consequence in terms of time and manpower expenditures as well as other associated developmental costs. Further, given the productivity increases realized in the 4th generation language environment, their impact upon the Software Lifecycle and upon other traditional resource estimation measures becomes a critical issue warranting further analysis. It is this analysis and line of reasoning which are the topic of the remainder of this chapter.

3.2 THE PUTNAM MODEL IN THE 4TH GENERATION LANGUAGE ENVIRONMENT.

Empirical models, such as the Putnam Model, are by their

very nature short-lived in that they are developed and validated within the context of a particular technological environment. When the technological environment changes, empirical models must be revalidated within the context of the new technology. Throughout his works Putnam concedes as much with repeated references to the effects of changing state-of-the-art technology upon the formulas and relations developed. As technological advances such as 4th generation language systems are realized, it is incumbent upon the software engineering community to ascertain what impact these advances have upon traditional resource estimation techniques, if any. Critical to this analysis, however, is an immediate limitation which must be acknowledged. Recalling that the Putnam Model was based upon a vast body of empirical data collected on completed software development projects, it is obvious that current analysis is handicapped by the limited empirical data available concerning this relatively new technology of the 4th generation language systems. This limitation does not preclude analysis but does limit the validation of findings and assertions until more data is collected and experience is gained. With this in mind, the potential impact of 4th generation language systems upon the fundamental components of the Putnam Model may now be examined.

Recalling the Rayleigh/Norden Lifecycle Equation from Chapter 2:

$$Y' = 2Ka te^{-at^2}$$

and Putnam's derivation form for the (a) parameter:

$$a = 1/2 t_d^2$$

The impact of massive productivity increases gained from the use of a 4th generation language system would seem to fall principally upon two parameters of the lifecycle equation. Those being (a) the shape parameter of the curve governing time to peak manpower and (t_d) the elapsed time from the start of cycle parameter. It is obvious that productivity increases on the order of 20:1 and higher would dramatically alter either of these parameters since both represent or imply some function of elapsed time in the developmental process. The effect of permitting the (a) parameter to vary while maintaining the (K), or manpower, parameter constant was previously demonstrated and portrayed. This portrayal is again presented as a review:

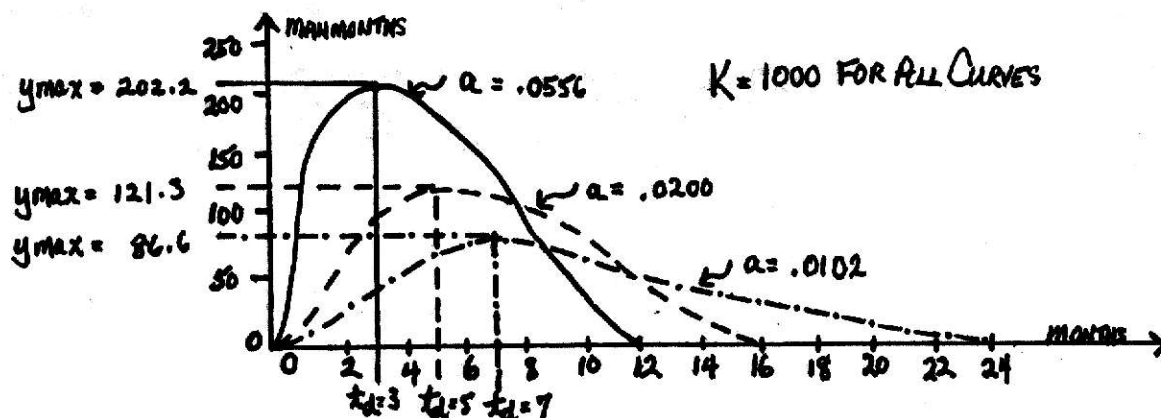


Figure 3.2.1 Lifecycle Curve with (K) constant. [Nor 63]

Recall that Putnam defined that (t_d) corresponded to the point in time at which peak manpower was reached. This was derived from the relation:

$$a = 1/2 (t_d^2)$$

This being true, it is obvious that this fundamental relation requires reevaluation in light of 4th generation productivity

increases since the (a), or shape of the curve parameter, will vary dramatically with the magnitude of the productivity increases realized. It is hypothetically possible that the relation itself may well remain valid within the context of the 4th generation language environment. However, it is equally feasible that, upon analysis of empirical data collected on projects using the new technology, a new relation may evolve or be required to be developed. A new relationship, if required, may well dramatically alter the characterization of the lifecycle equation. The precise image of the manpower utilization curve(s) in the 4th generation language environment cannot be determined in the absence of more empirical evidence and "data fitting" as Putnam chose to characterize the process. It is sufficient to note that any alteration of this fundamental relationship will have a significant impact upon the remaining relationships upon which Putnam's Software Equation is based.

The next relation, which logically could be expected to dramatically change in a 4th generation language environment, is that of Putnam's Difficulty Measure. Recall that this was portrayed as:

$$D = K/t_d^2$$

Of paramount importance in this analysis is Putnam's contention that as development time is shortened a dramatic increase in difficulty is to be expected. This was empirically validated by Putnam within the context of the technological environment in which it was developed - that being the 3rd generation high level

programming environment. However, within the context of the 4th generation language systems environment and associated dramatic productivity increases, there does not appear to be any logical basis upon which to make this assertion. This is due principally to the fact that within the 3rd generation environment one line of source code equalled one line of deliverable source code - obviously not valid within the context of 4th generation language systems. Recall that Putnam's Productivity Rate was portrayed as:

$$\overline{PR} = C_n D^{-2/3}$$

Where

(C_n) = a productivity constant

D = difficulty

Putnam acknowledged the impact of current state-of-the-art technology upon productivity as a measure of delivered source code. It is obvious that the major technological advances realized with the advent of 4th generation language systems must, by definition, have a direct impact upon this measure. There has been no suggestion in the literature researched to indicate a rise in difficulty for the programmer when using 4th generation language systems to generate the delivered code. In fact, assertions made in the literature are just the opposite - that indeed programming with 4th generation language systems is less complex and difficult than programming with traditional 3rd generation high level programming languages. Thus it is obvious that Putnam's Productivity Rate must be modified if only due to

the change in the productivity constant (C_n) realized by using a 4th generation language system. Indeed, further research and analysis may well indicate the need for an entirely new relation to portray the relationship between productivity and difficulty. Once again, significantly more analytical data is required in order to precisely determine the magnitude of change required in this fundamental relationship of the Putnam Model.

Having already called into question some fundamental relations upon which Putnam's Software Equation is based, it is logical to review the Software Equation within the context of the 4th generation language system environment. Recall Putnam's Software Equation:

$$S_s = 2.49 \cdot \bar{P}R \cdot K/6$$

$$= \frac{2.49}{6} C_n K^{1/3} t_d^{4/3}$$

The Generalized Form being:

$$S_s = C_K K^{1/3} t_d^{4/3}$$

Where

(C_K) subsumes $2.49/6$ (C_n) - A measure of the human/machine/state-of-the-art technology system.

It is not necessary to restate the obvious concerning the questions raised as to the validity of this relationship when analyzed within the context of the 4th generation language system environment. Previous discussion has led to the questioning of the empirical basis for both the (C_K) and (t_d)

relations. It is intuitively obvious that the Software Equation must be reevaluated within the context of this new technological environment. However, care must be taken to ensure that one significant point in this analysis is not overlooked - that point being the fundamental premise for Putnam's definition of delivered source code. In the traditional 3rd generation language environment, a descriptive definition for delivered source code can be portrayed as:

$$\text{SLOC} = \text{DLOC}$$

Where

SLOC = Lines of Source Code Written

DLOC = Delivered Lines of Source Code

While acknowledging that this portrayal is somewhat of an oversimplification, it nonetheless remains valid in terms of state-of-the-art technology in the 3rd generation language environment. Obviously, with productivity increases on the order of 20:1 and higher, this portrayal is not valid within the 4th generation language system environment. In fact, delivered lines of source code in the 4th generation language system environment is no longer a valid measure at all. It has become a moot point in the overall equation since productivity is based not upon delivered lines of source code but rather upon 4th generation systems code required to generate the deliverable product. If, for instance, it took 10 programmers eighteen months to produce 150,000 lines of deliverable application code in a traditional 3rd generation applications environment, those same 10

programmers can now write 7500 lines of 4th generation language source code, thereby generating 150,000 lines of deliverable application code, in 9.37 months, assuming a productivity increase of 20:1 and all other factors remaining equal. It is therefore obvious that the Putnam Model, in its current form without revalidation within the context of the new technology, cannot be used to estimate 4th generation language system projects.

The potential impact of 4th generation technological advances upon traditional resource estimation tools has been clearly demonstrated. The need for collection of empirical data in this new environment as well as for continued research into the validation of fundamental relationships comprising the Software Lifecycle in this new environment should be obvious to all. The potential impact of this new technology upon traditional software estimation tools cannot be overstated nor overlooked by the software engineering community. A summarization of the purpose of this report and recommendations for future research is the focus of Chapter 5. However, before moving to this topic, a brief discussion of an automated tool developed to graphically portray the potential impact of this new technology upon the behavior of the traditional Software Lifecycle Curves is in order. The general area of an automated tool to assist in the portrayal of this phenomenon is the topic of Chapter 4.

CHAPTER 4 GRAPHIC PORTRAYAL OF THE IMPACT OF 4TH GENERATION TECHNOLOGY UPON THE PUTNAM MODEL.

4.1 BACKGROUND. As an adjunct to the research and analysis which form the crux of this report, a brief Turbo Pascal program was written which permits the graphic portrayal of the behavior of the Software Lifecycle Curves when certain parameters are varied in view of the technological gains. The source code for this program is contained in Appendix A of this report. It is important to note that this program was developed for a very limited purpose - that being graphic representation of the Software Lifecycle Curves. It is not intended as a predictive tool and it suffers the additional limitation of being developed on a system with no graphics capability. Due to these factors as well as the limitations placed upon the development by the language itself, the program is extremely limited in scope. It was necessary to linearize equations, add scaling factors, and reformat equations in order to "trick" Turbo Pascal into performing some of the calculations required and to do the plotting required for curve representation.

The tool basically accomplishes three functions: 1) It plots the traditional Rayleigh/Norden Curve. 2) By permitting the user to vary the time or shape parameters, it plots the "compressed" Rayleigh/Norden Curve. 3) By applying a range of productivity increase values to the equation, the developmental portion of the Software Lifecycle may be portrayed. However, due to the limited graphics capability and scaling factors required

to portray this curve on the developmental system, a word of caution is required. The program is designed to provide an "overlayed" pictorial representation of the three curves simultaneously. In doing so, and due again to the limitations of the developmental system, the developmental curve when applying productivity constants to the equation appears to compress to a point where it exceeds the y-max of the original curve. This, in fact, does not happen and is merely a function of experimentation with the scaling factor in order to permit graphic portrayal of the three curves simultaneously in order to give the user a full appreciation of the dramatic effects the new technology has upon the traditional curve's performance.

4.2 HIERARCHICAL REPRESENTATION OF PROGRAM MODULES.

The tool, as developed, is comprised of four functional modules which are embodied in one program segment. These modules are titled GET_PARAMETER, CALC_RN_CURVE, CALC_DC_CURVE, and TERMINAL_DISPLAY. The following Hierarchy Diagram displays the functional relationship of the modules within the program:

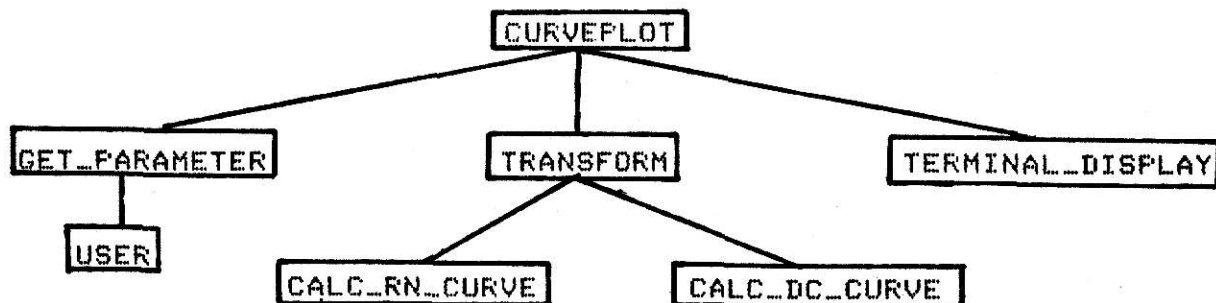


Figure 4.2.1 Program Curveplot Hierarchy Diagram

4.3 MODULE SPECIFICATIONS FOR PROGRAM CURVEPLOT.

The module specifications for each of the four primary program modules will be detailed in this section. The format utilized will be the module name, a short description of the module's tasks, calling sequence, the module's position within the hierarchy diagram, files or global data items accessed or changed, error messages produced, any unique module behavior, inputs, and outputs.

4.3.1 MODULE GET_PARAMETER.

Specification for Module: GET_PARAMETER

Short Description of Module's Tasks:

This module gets parameter data based on user keyboard inputs.

Calling Sequence: Turbo Pascal Commands - get(K)
 readln(More_work)
 readln(PR)

Position in Hierarchy Diagram

Modules that call this: CALC_RN_CURVE, CALC_DC_CURVE

Modules that are called: None

Files or Global Data Items accessed or changed: None

Error Messages Produced: None

Any externally noticeable behavior not previously described:

None

INPUTS

| Names | Description | Type | Domain |
|-------|-------------|------|--------|
|-------|-------------|------|--------|

| K | holds user defined value for curve shape parameter | real | 500 - 1500 |
|---------|--|------|--------------------------------|
| PR | holds user defined value for productivity | real | Implementation dependent range |
| OUTPUTS | | | |
| Names | Description | Type | Domain |
| K | User defined value selected | real | 500 - 1500 |
| PR | User defined value | real | Implementation dependent range |

4.3.2 MODULE CALC_RN_CURVE

Specification for Module: CALC_RN_CURVE

Short Description of Module's Tasks:

This module's two principal tasks are to calculate and plot the traditional Rayleigh/Norden Manpower Utilization Curve as well as an accelerated version of that curve assuming a productivity increase factor

Calling Sequence: while work_flag do

Position in the Hierarchy Diagram

Modules that call this: None

Modules that are called: TERMINAL_DISPLAY

Files or Global Data Items accessed or changed:

a - a fixed shape parameter for the curve

max_prog_time - an arbitrary maximum project length

Error Messages Produced: None

Any externally noticeable behavior not previously described:

None

INPUTS

| Names | Description | Type | Domain |
|-----------|---|---------|---------------------------|
| K | User defined input value for shape parameter of curve | real | 500 - 1500 |
| a | a fixed shape parameter for the curve | real | 0.02 |
| t | elapsed time | real | 0 - 26 |
| more_work | holds input to terminal display | char | 'y' or 'Y' |
| C | holds user input to terminal display | char | 'C' or ' ' |
| y_hat | holds value to represent point in time at which peak effort is utilized | real | machine - dependent range |
| work_flag | holds input to work status | boolean | T or F |

OUTPUTS

| Names | Description | Type | Domain |
|---------|--|------|---------|
| (. , *) | character representation of xy coordinate sent to terminal display | char | '.' '*' |

4.3.3 MODULE CALC_DC_CURVE

Specification for Module: CALC_DC_CURVE

Short Description of the Module's Tasks:

This module's sole purpose is to calculate and plot the Development and Coding portion of the overall Rayleigh/Norden Manpower Utilization Curve.

Calling Sequence: while work_flag do

Position in the Hierarchy Diagram

Modules that call this: None

Modules that are called: Terminal_Display

Files or Global Data Items accessed or changed:

a - a fixed shape parameter for the curve

max_proj_time - an arbitrary maximum project length

INPUTS

| Names | Description | Type | Domain |
|-----------|--|---------|-------------------------|
| PR | User defined value to represent programmer productivity | real | machine dependent range |
| K | User defined input value for shape parameter of the curve | real | 500 - 1500 |
| a | A fixed shape parameter for the curve | real | 0.02 |
| td | A calculated value for td or elapsed time from start of project | real | machine dependent range |
| C | Holds user input to terminal prompt | char | 'Y' or 'y' |
| more_work | Holds user input to terminal prompt | char | 'C' or ' ' |
| work_flag | Holds user input to work status | boolean | T or F |
| y_hat_RN | Holds calculated value to represent point in time at which peak effort is reached in the Design & Coding portion of the Software Lifecycle | real | machine dependent range |

OUTPUTS

| Names | Description | Type | Domain |
|-------|-------------|------|--------|
|-------|-------------|------|--------|

| | | | |
|--------|--|------|------|
| (\$) | Character representation of xy coordinate sent to terminal display | char | '\$' |
|--------|--|------|------|

4.3.4 MODULE TERMINAL_DISPLAY

Specification for Module: TERMINAL_DISPLAY

Short Description of Module's Tasks:

This module's sole purpose is to output program results to the terminal display

Calling Sequence: write('.', '*', '\$')

Position in the Hierarchy Diagram

Modules that call this: CALC_RN_CURVE
CALC_DC_CURVE

Modules that are called: None

Files or Global Data Items accessed or changed: None

Error Messages Produced: None

Any Externally Noticeable Behavior not previously described:

Based upon the values calculated within the formulas, when the curves are plotted the upper range of points plotted may exceed the capability of the terminal display. Additionally, the Design & Coding curve appears to exceed the limits of the calculated Rayleigh/Norden Manpower Utilization Curves when certain combinations of variable values are used. This is not the case. It is merely an implementation limitation upon graphic portrayal.

INPUTS

| Name | Description | Type | Domain |
|-------------|---|------|----------------|
| (. , *, \$) | Character representation of xy coordinate position | char | '.', '*', '\$' |

OUTPUTS

None

4.4 PSEUDO-CODE FOR PROGRAM CURVEPLOT

This section provides an English-like, pseudo-code version of the program logic upon which Program Curveplot is based. While not in the formal syntax of pseudo-code, the author's program logic is easily recognizable in the following form.

Program Curveplot

Variables:

Global:

- a - fixed number 0.02 as a shape parameter for the curve
- max_proj_time - fixed number 26 indicating maximum project length

Local:

- more_work - variable indicating user-defined course of action
- C - variable indicating user-defined course of action
- t - variable representing time along x-axis
- td - variable representing a point in time along x-axis
- y_hat - variable representing a point in time along y-axis
- y_hat_RN - variable representing a point in time along y-axis

```

K          - variable representing man-months of effort
PR         - variable representing average programmer
             productivity rate
work_flag  - boolean switch indicating work status

start

set work_flag to True
while work_flag do
    prompt user for curve shape parameter
    read value
    set time = 0.0
    clear screen
    while t < max_proj_size do
        calculate y_hat expression
        if t < 1 do
            calculate y_hat expression RN
        write '.' at pre-determined coordinate position
        if t < 1 do
            calculate y_hat expression accelerated RN
        write '*' at pre-determined coordinate position
        increment t = t + 2.0
        prompt user for any parameter changes
        read response
        if response (more_work) = 'y' or 'Y' then
            work_flag = True
        else work_flag = False
    end; {while t < max_proj_time loop}
    prompt user for desired PR value

```

```

read PR value
calculate value for td
write td value

read continuation parameter
reset work_flag
while work_flag = true do
    set t = 0.0
    ClrScr
        while t <= max_proj_time do
            calculate RN Manpower Util Curve
            calculate accel MP Util Curve
            write '.' RN Manpower Curve
            write '*' accel RN Curve
            write '$' D/C portion of standard
                R/N Manpower Curve

            set t = t + 2.0
        end while t < max_proj_time do
        prompt user for any parameter changes
        read changes
        if no change exit, if change loop

    end while work_flag do
end. program curveplot

```

This marks the end of the automatable tool description. While realizing that this particular application is limited in scope, nevertheless it serves a useful function in terms of

graphic portrayal of savings realized in the 4th generation language systems environment. With this explanation completed, it is appropriate to proceed to Chapter 5 for the summarization of the purpose of this report and the recommendations for future research in this area.

CHAPTER 5 A SUMMARY OF CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH.

5.1 OVERVIEW.

The purpose for undertaking this analysis of the Putnam Model within the context of current technological advances was two-fold: 1) to demonstrate the historical basis for and the validity of the Putnam Model within the context of the environment in which it was developed and 2) to highlight the requirement for, and hopefully generate academic interest in, a revalidation of traditional resource estimation models within the context of 4th generation language systems technological advances. A brief summary of the author's conclusions and recommendations for future research in this area are the subject of the remainder of this report.

5.2 CONCLUSIONS.

The author admits to a certain predisposition when this effort was initiated - that predisposition being that the technological advances realized within the 4th generation language systems environment as well as the rapid growth of "artificially intelligent" systems seemed to portend the immediate demise of traditional software development resource estimation models. However, after fully exploring the fundamental relations upon which the Putnam Model is based, this portent of doom was found to be completely premature. Certainly

a reevaluation and revalidation of the models is essential to our understanding of the relationships between technological advances and traditional software estimation models fundamental bases. However, it would be presumptuous and premature to bias one's view prior to the completion of this process. It seems logical, and indeed probable, that the weighting factors and scaling factors derived in the formulation of Putnam's original model will indeed require modification in view of productivity increases realized in the 4th generation language systems environment. However, the magnitude of change(s) required will be determined and validated only upon completion of a massive data collection effort and a complete analysis of software development projects using this new technological environment. Certainly Putnam's original methodology remains fundamentally sound and future researchers would be wise to emulate his step-wise refinement or building-block formal approach in future validation efforts. It is the author's conclusion that Putnam's Model will require modification, specifically in terms of power factors used in the original formulas, in order to remain a viable tool within the context of the 4th generation language systems environment.

5.3 RECOMMENDATIONS.

It is hoped that the premise for and the arguments presented in this report will generate academic interest in a thorough reevaluation not only of the Putnam Model but also of other

resource estimation models as well in light of the technological advances realized with the advent of 4th generation language systems. The collection of empirical data upon which to base this reevaluation will be a major undertaking but is absolutely essential if resource estimation models currently in use are to remain viable. The credibility of the Software Engineering discipline within the business community is clearly at stake. Indeed, the economic viability of many software development companies may well depend upon the validity and utility of the software development resource estimation tools available to them. The time for a coordinated, complete reevaluation of the available resource estimation tools has arrived. The Software Engineering community must not delay undertaking this vital effort. Our academic credibility as a discipline and indeed, the economic viability of many software development companies, may well depend upon the successful completion of this requirement.

APPENDIX A

This appendix is the source code for program curveplot. Note - while the code is complete and accurate, the user must follow normal turbo pascal conventions for comments, continuation of statements from one line to another, and capitalization of identifiers. The code listed below is technically accurate; however in order to fulfill the administrative requirements of this report editing shortcuts were employed for formatting purposes.

```
program curveplot (input, output);

const
    a = 0.02;                {a fixed shape parameter for the
                             curves}
    max_proj_time = 26;      {an arbitrary maximum project
                             length in man-months of effort}

var
    more_work, C : char;    {variables indicating user-
                             defined courses of action.}
    t : real;               {time variable along x-axis}
    y_hat,
    y_hat_RN : real;        {point in time along y-axis at
                             which peak manpower is reached.}
    K : real;               {total man-months of effort}
    work_flag : boolean;    {boolean switch indicating
                             actions.}
                             {average productivity rate per
                             month.}
    td : real;              {elapsed time to reach peak
                             effort.}

begin
    work_flag := True;
    while work_flag do
        begin
            writeln ('What value for K do you want
                     (500,700,1000,1500)? ');
```

```

readln (K);
t := 0.0;
ClrScr;
while t <= max_proj_time do
  begin
    {This section of code portrays the traditional
    Rayleigh/Norden Manpower Utilization Curve.}

    y_hat := 2 * K * a * t * Exp(-a * Sqr(t));
    if t < 1 then
      Gotoxy(1, 24 - Trunc(0.075 * y_hat))
    else
      {Using 3 as a multiplier for pixel per character width}
      Gotoxy(Trunc(3 * t), 24 - Trunc(0.075 * y_hat));
    write(' ');

    {This section of code portrays an accelerated version
    of the Rayleigh/Norden Manpower Curve assuming an
    average productivity increase of 1:5 or .2 as an
    example, and multiplying by a scaling factor of 10 to
    keep the curve from narrowing too fast for pictorial
    representation.}

    if t < 1 then
      Gotoxy(1, 24 - Trunc(0.075 * y_hat))
    else
      Gotoxy(Trunc(0.2 * Trunc(10 * t)), 24 - Trunc(0.075 *
      y_hat));
    write('*');
    t := t + 2.0;
  end;
writeln;

  {This section of code allows the user to vary any
  parameters used thus far to visualize the effects of
  altering these parameters on the representation of the
  curves.}

  writeln ('Do you want to change any other parameters
  (Y/N)?');
  readln (more_work);
  if (more_work = 'y') or (more_work = 'Y') then
    work_flag := True
  else
    work_flag := False;
end;

```

{This section of code permits the user to alter the PR value which leads to the calculation of a user-defined value for td which in turn allows the visual representation of the Putnam's Design & Coding portion of the overall Software Lifecycle.}

```
writeln ('What value for PR do you want (855.40736) ?');
readln (PR);
```

{Utilizing the formulas presented in Putnam's Paper entitled "General Empirical Solution ..." to determine the Software Equation: $S_s := 2.49 \text{ div } PR * 1200 \text{ Cubrt } K * 4/3 \text{rt } td$ as well as "massaging" this formula and Putnam's formulas - $PR = Cn * D^{** -2/3}$ and $D = K/td(td^{** 2})$ a new linearized expression utilizing the in-built functions of Turbo Pascal is derived permitting the calculation of td. }

```
td := exp(ln(exp(ln(k) * 2/3) * (PR/12000)) * 3/4);
writeln ('The current value of td is (3.65): ',td);
repeat
  writeln ('Push /space-bar/ and return to Continue');
  readln (C);
until C = ' ';
ClrScr;
```

{The last section of code builds upon the calculations performed thus far, calculations for the Design & Coding portion of the Software Lifecycle is based upon the User provided input, and handles the remaining program I/O.}

```
work_flag := True;
while work_flag do
  begin
    t := 0.0;
    ClrScr;
    while t <= max_proj_time do
      begin

        {Representation of the "Stand-alone" manpower
        utilization curve is now calculated.}

        y_hat := 2 * K * a * t * Exp(-a * Sqr(t));
```

```
{Utilizing the value for td calculated earlier, we
an adjusted Rayleigh/Norden Curve.}
```

```
y_hat_RN := (K/Sqr(td)) * t * Exp(-0.5 *
Sqr(t)/Sqr(td));
```

```
{Plot the normal manpower utilization curve.}
```

```
if t < 1 then
  Gotoxy(1, (24 - Trunc(0.075 * y_hat)))
else
  Gotoxy(Trunc(3 * t), (24 - Trunc(0.075 *
y_hat)));
write('.');
```

```
{Plot the accelerated (1:5 or better) M/U Curve. }
```

```
if t < 1 then
  Gotoxy(1, (24 - Trunc(0.075 * y_hat)))
else
  Gotoxy(Trunc(0.02 * Trunc(10 * t)),
(24 - Trunc(0.075 * y_hat)));
write('*');
```

```
{Plot the Rayleigh/Norden curve for Development - }
{Coding Section of the Software Lifecycle. }
```

```
if t < 1 then
  Gotoxy(1, 24 - Trunc(0.075 * y_hat_RN))
else
  Gotoxy(Trunc(0.2 * Trunc(10 * t)),
24 - Trunc(0.075 * y_hat_RN));
write('$');
t := t + 2.0;
```

```
end;
writeln;
```

```
writeln('Do you want to change any other parameters
Y/N)?');
```

```
readln (more_work);
```

```
if (more_work = 'y') or (more_work = 'Y') then
```

```
  work_flag := True
```

```
else
```

```
  work_flag := False;
```

```
end;
```

```
end.
```

BIBLIOGRAPHY

Bailey, John W. and Basili, Victor R., "A Meta-Model for Software Development and Resource Expenditures," Proceedings of the 5th International Conference on Software Engineering, IEEE, New York, 1981.

Basili, Victor R. and Reiter, Robert W., Jr., "Evaluating Automatable Measures of Software Development," Workshop on Quantitative Software Models for Reliability, Complexity, and Cost, IEEE, New York, 1979.

Basili, Victor R., "Quantitative Software Complexity Models: A Panel Summary," Workshop on Quantitative Software Models for Reliability, Complexity, and Cost, IEEE, New York, 1979.

Basili, Victor R., "Tutorial on Models and Metrics for Software Management and Engineering," Computer Society Press, 1980.

Boehm, Barry W., Software Engineering Economics, Prentice-Hall, 1981.

Brooks, Fredrick P., Jr., The Mythical Man-Month, Addison-Wesley, 1975.

Daly, Edmund B., "Management of Software Development," IEEE Transactions on Software Engineering, May 1977, IEEE, New York 1977.

DACS, Data Analysis Center for Software, "Software Engineering Research Review, Quantitative Models," Computer Sciences Corporation, 1979.

DeMarco, Tom, Controlling Software Projects, Management, Measurement & Estimation, Yourden Press, 1982.

Martin, James A., Application Development Without Programmers, Prentice-Hall, 1982.

Martin, James A. and McClure, Carmo, Software Maintenance, The Problem and Its Solutions, Prentice-Hall, 1983.

Martin, James A. and McClure, Carmo, Action Diagrams, Clearly Structured Program Design, Prentice-Hall, 1985.

Norden, Peter V., "Useful Tools for Project Management," Operations Research and Development, edited by B. V. Dean, John Wiley & Sons, 1963.

Norden, Peter V., "Project Lifecycle Modelling: Background and Application of Lifecycle Curves," First Software Lifecycle Workshop, 1977.

Pressman, Roger S., Software Engineering, A Practicioners Approach, McGraw-Hill, 1982.

Putnam, Lawrence H., "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Transactions on Software Engineering, July 1978, IEEE, New York, 1978.

Putnam, Lawrence H., "Example of an Early Sizing, Cost and Schedule Estimate for an Application Software System," Proceedings, COMESAC 28, IEEE, New York, 1978.

Putnam, Lawrence H., "Software Costing and Lifecycle Control," Workshop on Qualitative Software Models and Cost, IEEE, New York, 1979.

Putnam, Lawrence H. and Fitzsimmons, Ann, "Estimating Software Costs," Datamation, September 1979.

Putnam, Lawrence H. and Fitzsimmons, Ann, "Estimating Software Costs," Datamation, October 1979.

Putnam, Lawrence H. and Fitzsimmons, Ann, "Estimating Software Costs," Datamation, November 1979.

Putnam, Lawrence H., Tutorial on Software Cost Estimating and Lifecycle Control: Getting the Software Numbers, Computer Society Press, 1980.

Rudolf, Eberhard E., "Productivity in Computer Application Development," Department of Mangagment Studies Working Paper No. 2, University of Auckland, March 1983.

Vick, C. R. and Ramamoorthy, C.V., Handbook on Software Engineering, Van Nostrand Reinhold, 1984.

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

Walston, C.E., "Working Group on Software Cost," Workshop on Qualitative Software Models and Cost, IEEE, New York, 1979.

SOFTWARE DEVELOPMENT RESOURCE ESTIMATION
IN THE 4TH GENERATION ENVIRONMENT

by

DANIEL RAYMOND MOYER

B. S., Bowling Green State University, 1971

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

ABSTRACT

One of the most important tasks performed by software engineers is that of developing useful models and equations which permit the prediction of project costs for software development projects. Traditional models currently in use were developed and validated within the context of existing 3rd generation state-of-the-art technology. The advent and widespread use of code generators and 4th generation language systems realizing productivity increases on the order of 20:1 and higher mandates a reevaluation and revalidation of traditional models within the context of this new technological environment.

This report presents an examination of the formal foundations of the Putnam Model and presents the potential impact of 4th generation language systems upon this traditional model. The requirement for revalidation of traditional models within the context of this new technological environment is also presented.