# 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: David A. Gustafson

Major Professor

LD
2668
.R4
1986
M63
c.2

A11202 663944

# TABLE OF CONTENTS

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 succint 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 requirments, 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 forseeable 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 $(E = 3.4 + .72 * DL^{1.17})$ | 15319 | 226534 | 509724 |
| Boehm $(MM = 2.4(KDSI)^{1.05})$ | 18371 | 206125 | 426787 |
| Walston & Felix $(E = 5.2\ L^{.91})$ | 12080 | 98190 | 184503 |

Table 1.3.1 Comparitve Metric Values

The principal reason for the ommision 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 practicioner 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:

$$Y' = 2Kate^{-at^2}$$

Where

y′ = manpowere 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.

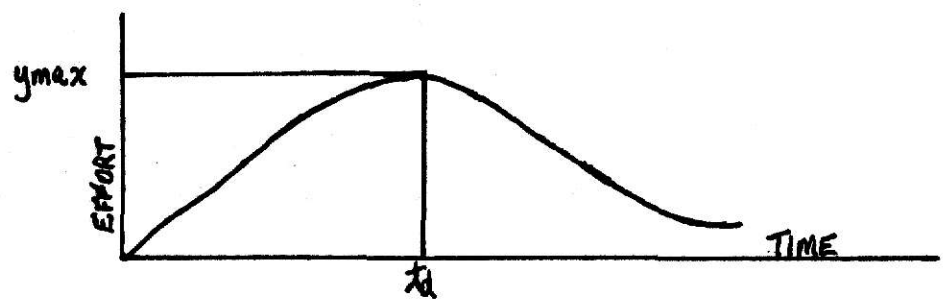


Figure 2.2.1 Rayleigh/Norden Manpower Utilization Curve

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 or the ($t_d$) constant and permitting the (K) value to vary.

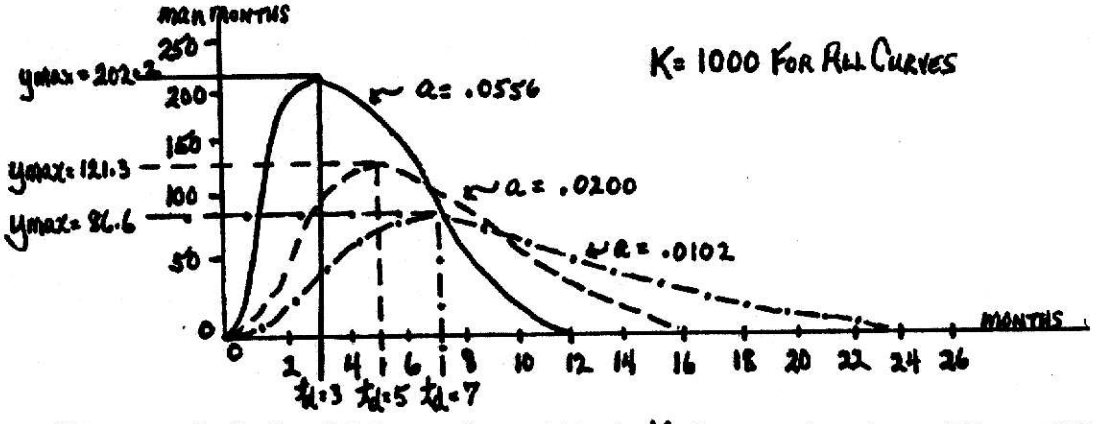


Figure 2.2.2 Lifecycle with ( K ) constant. [Nor 63]

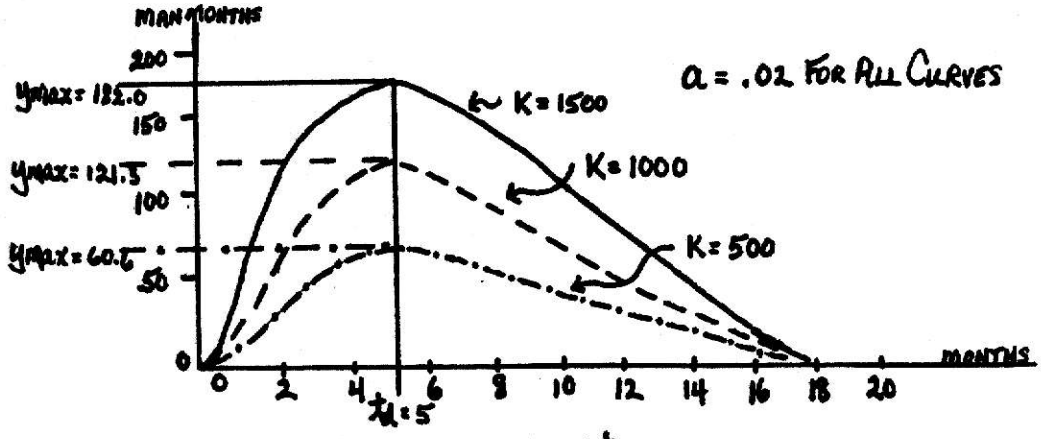


Figure 2.2.3 Lifecycle with ($t_d$) constant. [Nor 63]

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 Rayliegh/Norden Curve via linearization in the form:

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

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:

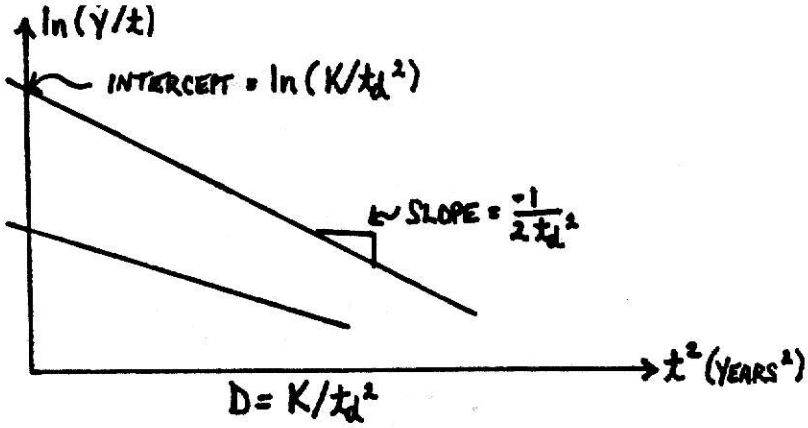


Figure 2.3.1 Putnam's Difficulty Measure

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 evalution 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 = Sclaing 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 pictoral representation of Putnam's Software Lifecycle follows:

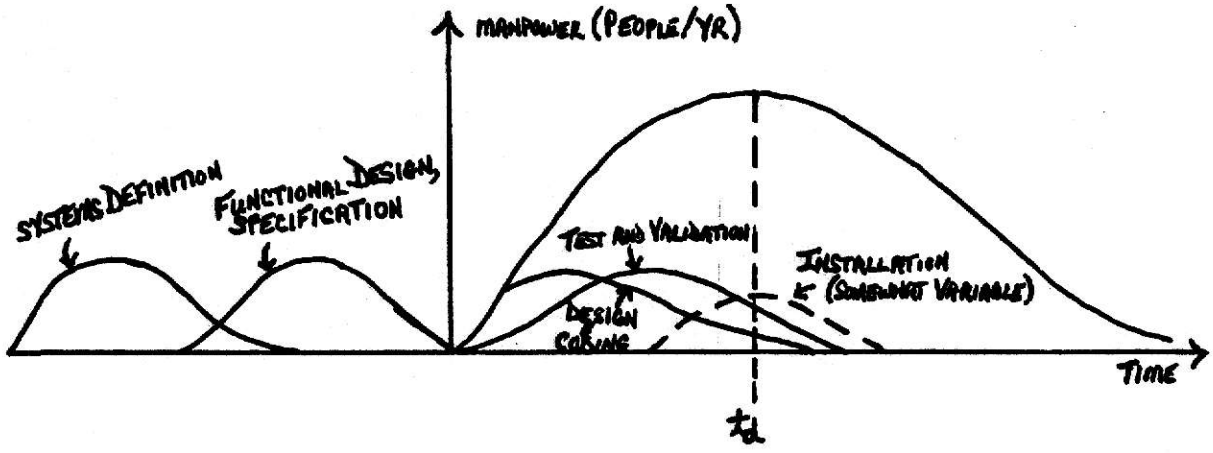


Figure 2.3.2 Putnam's Software Lifecycle

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