Quantitative Assessment Of Software Development Project Management Issues Using Process Simulation With System Dynamics Elements

2006

Carolyn Mizell
University of Central Florida

Find similar works at: http://stars.library.ucf.edu/etd

University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation

http://stars.library.ucf.edu/etd/995

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
QUANTITATIVE ASSESSMENT OF SOFTWARE DEVELOPMENT PROJECT
MANAGEMENT ISSUES USING PROCESS SIMULATION MODELING WITH SYSTEM
DYNAMICS ELEMENTS

by

CAROLYN BARRETT MIZELL
B.S. Clemson University, 1989
M.S. University of Central Florida, 1994

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Industrial Engineering and Management Systems
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Major Professor: Dr. Linda Malone
Summer Term
2006
ABSTRACT

The complexity of software development projects makes estimation and management very difficult. There is a need for improved cost estimation methods and new models of lifecycle processes other than the common waterfall process. This work has developed a new simulation model of the spiral development lifecycle as well as an approach for using simulation for cost and schedule estimation. The goal is to provide a tool that can analyze the effects of a spiral development process as well as a tool that illustrates the difficulties management faces in forecasting budgets at the beginning of a project which may encourage more realistic approaches to budgetary planning.

A new discrete event process model of the incremental spiral development lifecycle approach was developed in order to analyze the effects this development approach has on the estimation process as well as on the cost and schedule for a project. The input data for the key variables of size, productivity, and defect injection rates in the model was based on analysis of Software Engineering Laboratory data which served as the basis for developing probability distributions for key variables to enable analysis of the effects of uncertainty in early project estimates. In addition, the benefits of combining a separate system dynamics model with a discrete event process model were demonstrated.

This work includes a major case study of a cancelled NASA software development project that experienced cost and schedule problems throughout its history. Analysis was performed using stochastic simulation with derived probability distributions for key software development factors. A system dynamics model of human resource issues was also combined with the process model to more thoroughly analyze the effects of turnover on a project. This
research has demonstrated the benefits of using a simulation model when estimating to allow for more realistic budget and schedule determination including an interval estimate to help focus on the uncertainty of the estimates.
I dedicate this work to my mother, Marilyn Barrett, who passed away before it was completed. She was always in my thoughts as I worked to finish this and I truly hope that she would be proud of it.
ACKNOWLEDGEMENTS

I would like to first and foremost acknowledge my gratitude for the support and love provided by my family and friends throughout this long and challenging process. My husband, Richard and my son, Matthew, had to live with me through all of the trials and tribulations that go with completing a dissertation and I feel truly blessed to have them in my corner. Much thanks and appreciation goes to my parents for setting me on this path and for always believing in me.

My sincerest thanks must also go to my advisor, Dr. Linda Malone, for being such a wonderful mentor whose guidance and friendship were so helpful to me throughout this period of my life. I would also like to thank the rest of my committee, Dr. David Raffo, Dr. Renee Butler, Dr. Mansooreh Mollaghasemi, and Dr. Luis Rabelo for their support and leadership.

I must acknowledge the great opportunity afforded to me by NASA in the form of a graduate fellowship. I would especially like to thank Jim Heald and Rita Willcoxon for their support in obtaining the Kennedy Space Center Graduate Fellowship. This fellowship provided the best means for me to perform this work and to obtain this degree. I am especially grateful to the support and encouragement given to me by my supervisor, J. David Collins, and my director, Oscar Toledo. David Collins is a supervisor of the highest standards who has been a mentor to me for many years. I deeply value his opinion and advice and he has always managed to help me overcome the many challenges I have faced. I feel truly blessed to work for such a fantastic manager and for an agency that values education. My hope is that my research will be useful to NASA.
I would also like to acknowledge the tremendous help I received in obtaining data on the Checkout and Launch Control System project. CLCS project team members eagerly provided data and project information, even though the project had been cancelled for several years and it was necessary to go back through old notes and data. I was truly lucky to have access to so much pertinent data for a project that provides such a great opportunity to learn. I would especially like to thank Larry Wilhelm, Richard Smith, Mike Bolger, Ben Bryant, and Jo Gonzales.

I received excellent modeling and programming support from Charles Curley and Umanath Nayak. They are top programmers and I am so appreciative of their work and their friendship. I would finally like to thank fellow NASA employees, Martin Steele, Chuck Lostroscio, and Keith Britton for their support and help with this endeavor.
# TABLE OF CONTENTS

LIST OF FIGURES ....................................................................................................................... xi
LIST OF TABLES ............................................................................................................................ xiii
LIST OF ACRONYMS/ABBREVIATIONS .................................................................................. i
CHAPTER ONE: INTRODUCTION ............................................................................................. 1
   Background ................................................................................................................................. 3
   Statement of Problem ............................................................................................................... 5
   Research Questions .................................................................................................................. 7
CHAPTER TWO: LITERATURE REVIEW ................................................................................... 11
   Software Engineering .............................................................................................................. 11
   Software Development Project Management ....................................................................... 18
   Models ....................................................................................................................................... 27
   Computer Simulation ............................................................................................................. 30
   Software System Dynamics ................................................................................................... 31
   Process Models ....................................................................................................................... 34
CHAPTER THREE: METHODOLOGY ......................................................................................... 39
   Process Model Enhancements ............................................................................................... 42
   Model Inputs ............................................................................................................................ 48
   Model Output .......................................................................................................................... 54
   Addition of System Dynamics ............................................................................................... 56
   Incremental Spiral Development Model .............................................................................. 60
   Verification and Validation of Model ...................................................................................... 62
# LIST OF FIGURES

Figure 1: Spiral Development Model (Boehm 1988) ................................................................. 14
Figure 2: Summary of Software Development Modeling Work .................................................. 38
Figure 3: PATT Process Model .................................................................................................. 43
Figure 4: Spiral Development Model ........................................................................................ 45
Figure 5: Research Approach...................................................................................................... 46
Figure 6: Software costing and sizing accuracy versus phase .................................................... 49
Figure 7: Under-Estimating Size According to Phase (Adapted from Boehm, Abts et al. 2000) 50
Figure 8: Process Model Main Screen Output Display ................................................................. 54
Figure 9: System Dynamics Human Resource Model ................................................................. 58
Figure 10: Flowchart of Process to Combine Models ................................................................. 59
Figure 11: Use of PATT for Spiral Model .................................................................................. 60
Figure 12: Possible Under-estimation of Size per Phase ............................................................ 72
Figure 13: Sample PATT Output Data ....................................................................................... 76
Figure 14: Example Data from Five Replications ..................................................................... 77
Figure 15: Human Resource Model in Vensim ......................................................................... 94
Figure 16: Turnover vs. Time (Baserun) ................................................................................... 98
Figure 17: Assimilation Delay vs. Time (Baserun) .................................................................... 99
Figure 18: Number of Inexperienced Personnel vs. Time (Baserun) ......................................... 100
Figure 19: Number Experienced Personnel vs. Time (Baserun) ............................................... 101
Figure 20: Turnover vs. Time (Experiment, Baserun) ............................................................... 102
Figure 21: Assimilation Delay vs. Time (Experiment, Baserun) ................................................ 103
Figure 22: Number of Inexperienced Staff vs. Time (Experiment, Baserun) .............................. 104
Figure 23: Number of Experienced Staff vs. Time (Experienced, Baserun) ............................. 105
Figure 24: Spiral Development Model ....................................................................................... 108
Figure 25: Waterfall Process Simulation Model ......................................................................... 111
Figure 26: Use of PATT for Spiral Model .................................................................................. 112
LIST OF TABLES

Table 1: SeL Productivity Data (LOC/Hr) Summary ................................................................. 51
Table 2: SEL Defect Data (Defects/KSLOC) Summary ........................................................... 52
Table 3: Probability Distributions for Defect Data ................................................................. 53
Table 4: Percentage of Effort by Phase .................................................................................. 61
Table 5: Defect Injection Probability Distributions ............................................................... 81
Table 6: Results from 150 Replications ............................................................................... 83
Table 7: Results Using Different Probability Distributions .................................................. 87
Table 8: Summary of Effects of Turnover Rates ................................................................. 106
Table 9: Percentage of Effort by Phase for Waterfall and Spiral Processes ....................... 113
Table 10: Defect Injection Rates by Phase ......................................................................... 114
Table 11: Size Estimate Per Increment ............................................................................. 116
Table 12: Comparison of Outputs for Spiral and Waterfall Models .................................... 117
Table 13: Incremental Size Distributions ......................................................................... 118
Table 14: Size Estimates for Each Increment .................................................................. 128
Table 15: Input Values for Productivity and Defects ......................................................... 129
Table 16: Results for Waterfall Model With No Size Uncertainty ...................................... 130
Table 17: Results for Waterfall Model with Size Uncertainty ............................................ 130
Table 18: Model Results of Incremental Model With No Size Uncertainty ...................... 131
Table 19: Model Results of Incremental Model with Size Uncertainty .............................. 131
Table 20: Durations for Each Increment with Size Uncertainty ....................................... 132
Table 21: Actual Increment Sizes ............................................................................................... 134

Table 22: Interval Estimates at Different Points in the Project .................................................. 135
# LIST OF ACRONYMS/ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CeBase</td>
<td>Center for Empirically Based Software Engineering</td>
</tr>
<tr>
<td>CMMI</td>
<td>Capability Maturity Model Integration</td>
</tr>
<tr>
<td>CLCS</td>
<td>Checkout Launch Control and Command System</td>
</tr>
<tr>
<td>COCOMO</td>
<td>Constructive Cost Model</td>
</tr>
<tr>
<td>IV&amp;V</td>
<td>Independent Verification &amp; Validation</td>
</tr>
<tr>
<td>KSLOC</td>
<td>Thousand Software Lines of Code</td>
</tr>
<tr>
<td>LOC</td>
<td>Lines of Code</td>
</tr>
<tr>
<td>LOC/Hr</td>
<td>Lines of Code per Hour</td>
</tr>
<tr>
<td>LPS</td>
<td>Launch Processing System</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
</tr>
<tr>
<td>PATT</td>
<td>Process Analysis Tradeoff Tool</td>
</tr>
<tr>
<td>QA</td>
<td>Quality Assurance</td>
</tr>
<tr>
<td>SEI</td>
<td>Software Engineering Institute</td>
</tr>
<tr>
<td>SEL</td>
<td>Software Engineering Laboratory</td>
</tr>
<tr>
<td>S.D.</td>
<td>Standard Deviation</td>
</tr>
</tbody>
</table>
CHAPTER ONE: INTRODUCTION

A Project Manager’s job can be facilitated if decision-making tools are available to predict the potential impact of decisions. Major decisions are made at the beginning of a software project to derive cost and schedule estimates, including the development life cycle to be followed and hiring and training practices. As a project progresses, a comparison of actual vs. planned may show the need to revise some elements of the initial planning or may indicate that there are problems or issues that need to be addressed. Additionally, changing managerial environments such as changing budgets and project deadlines can lead to unforeseen impacts. It is extremely important to quantify possible impacts to a software project as changes are taking place. A modeling tool that provides the probability of impact of potential project process changes or managerial decisions can be very useful to management in terms of understanding the possible ramifications of such changes and giving a quantitative look at which changes have the best chance of helping if incorporated.

Process simulation models, system dynamics models, and static cost models already exist for software development projects. Each of these tools has advantages and disadvantages and the appropriateness of each depends on the application. System dynamics models are useful tools for demonstrating the dynamic behavior of a project and are based on project variables and tasks as a whole with no process details or intricacies being captured. Process models, on the other hand, do provide great detail on the process and can be used to provide guidance on the sequence of process steps and information flows and can also be used to analyze proposed process changes. In addition, process models can support management planning and control
activities. This type of modeling, however, does not capture the interactions and structural relationships as effectively as system dynamics modeling.

Two key decisions a project manager must make before a project begins are how to derive an accurate cost estimate and which development lifecycle model to use. The accuracy of the cost estimate will impact whether or not a project can be successfully completed within budget and schedule. The development lifecycle model can affect the overall project success by driving the cost, schedule and quality of the software development.

Currently, process simulation models are used to evaluate cost and schedule impacts of process changes and system dynamics models are used to evaluate the effects of managerial decisions. A goal of this research is to demonstrate the effectiveness of using a combined simulation model for developing initial estimates as well as updating estimates throughout a project. The use of simulation in developing estimates will be compared to the present methods of creating cost and schedule estimates. In addition, there is a need to develop a simulation model of the spiral development life cycle because this newer life cycle model is gaining increasing popularity, especially for government and military software development projects. The effects of this type of development process on cost and schedule estimation will be investigated.

In addition to the above-mentioned areas of investigation there are other key issues that are of interest for the manager of a large, complex software development project. The following questions are of particular interest: What are the impacts of human resource issues such as turnover and experience levels? What are the impacts of requirements creep on cost and schedule estimates? These types of questions will be addressed by combining system dynamics
Background

Large operational systems software development projects, i.e. those that are performed by NASA, are difficult projects to manage. In general, software projects are difficult because the product is intangible and therefore, complex measures must be taken to adequately determine project status. According to the Standish Group’s study of success and failure rates of software development projects, less than one third of these projects: finish on time; are within budget; and deliver a product that performs to specifications (Brooks 1998).

The estimation for a software project’s cost and schedule, especially at the beginning of a project when there are many unknown factors, is extremely difficult. Therefore, estimates need to be updated throughout a project’s development life cycle. Also, estimates given in terms of ranges for cost and schedule with associated confidence intervals will make managers more aware of the uncertainty involved. As the project progresses through the software development life cycle, confidence in the estimate should get higher as more knowledge is obtained.

NASA has had success in the realm of large operational software projects such as the Space Shuttle Onboard Software project at the Johnson Space Center. This project is cited by the Software Engineering Institute as being level 5 on the Capability Maturity Model Integration Scale (CMMI), the highest possible on the widely-used process maturity scale (Carnegie Mellon University/Software Engineering Institute 1994). However, numerous large development projects have been cancelled due to large overruns and schedule delays. Numerous studies have
shown that the risk of major schedule delays, cost overruns, and cancellations increases as the size of the project increases and failures are more common than successes for large projects. A very large program is one which requires more than 100 programmers for a period of 5 or so years and results in a million source lines of code (Fairley 1985). The 1995 “Patterns of Software Systems Failures and Success” study concentrated on large projects since failures are more common than success for large systems (Jones 1998). The study found that the risk of cancellation or major delays rises rapidly as the overall size of the application increases. This study also found that the abilities of a team’s project management have a strong impact on the success of a project. The case study project that will be utilized for this research falls into the very large category.

There are different types of models that can be used to quantitatively model software development. Analytical models, such as COCOMO (Constructive Cost Model) (Boehm 1981) and SLIM (Putnam and Myers 1992), use mathematical relationships between variables for cost and project duration estimates. These types of models, however, ignore the dynamic aspects of project behavior and assume constant values for critical project factors such as productivity. The use of system dynamics and computer-based simulation has lead to other types of models. The most widely recognized system dynamics model for software development was developed by Abdel-Hamid and Madnick (Abdel-Hamid and Madnick 1991). This type of model provides for analysis of the interactions and relationships between related activities such as project management, code development, hiring, etc. The high-level nature of the model does not allow for capture of the details of the process or the relationships between development phases or initial project activities. Yet, these often have a great deal of influence on project success (Madachy 1994). Process models, on the other hand, capture the details of the process and allow
for the quantitative assessment of potential process changes and improvements, but are not as effective at capturing the dynamic interrelationships between social and technical factors as is system dynamics modeling. The Process Tradeoff Analysis (PTA) developed by Raffo can be used to quantitatively assess alternatives through use of discrete process simulation models of each process alternative (Raffo 1996).

**Statement of Problem**

Benchmark studies have shown that a majority of large software development projects run late, exceed their budgets, and are cancelled without ever reaching completion. It is difficult to control this type of project in terms of schedule, cost accuracy, and quality (Jones 2000). Since the 1970’s, the rigors of engineering have been applied to software development and this field is called software engineering. Project management is a critical aspect of software engineering that is still very challenging and often not given adequate priority. Project management consists of planning, scheduling, monitoring and controlling activities. Having more quantitative tools to help the project manager successfully perform these functions would be helpful. Project management capabilities have a strong influence on success and should be emphasized since there seem to be “many ways to fail and only a few ways to succeed” (Jones 1998).

The main sources of risk for software development projects deal with unrealistic schedules and budgets, misunderstood and continuously changing requirements, and lack of effective project management methodology (Jain and Dey 2004).

A Project Manager’s job can be facilitated if quantitative tools are available to predict the potential impact of decisions. Major decisions are made at the beginning of a software project to
derive cost and schedule estimates, including the development life cycle to be followed and hiring and training practices. As a project progresses, a comparison of actual vs. planned may show the need to revise any elements of the initial planning or may indicate that there are problems or issues that need to be addressed. Additionally, changing managerial environments such as changing budgets and schedules can lead to unforeseen impacts. It is extremely important to quantify possible impacts to a software project as changes are taking place. A modeling tool that provides the probability of impact of potential project process changes or managerial decisions can be very useful to management in terms of understanding the possible ramifications of such changes and giving a quantitative look at which changes have the best chance of helping if incorporated.

The goal of this research is to demonstrate the effectiveness of using a simulation model for creating and updating estimates as compared to other widely accepted methods. The development of effective cost and schedule estimates and the ability to update these continuously is still a weak link in software project management. Therefore, there is a need for improving the techniques used to develop estimates for software development projects. The developer of COCOMO, Barry Boehm, states that there is no guarantee that COCOMO can fit every organization’s style of development, data definitions, and set of operating assumptions (Boehm, Abts et al. 2000). As a matter of fact, improperly “tuned” COCOMO models have been shown to have an alarmingly high level of up to 600% inaccuracy (Kemerer 1987; Menzies and Sinsel 2000). Since simulation models can uniquely capture a wider range of project specifics that deal with particular development environments, this type of tool may provide added benefit for projects that are not typical.
The specific project management problems that will be investigated are the limitations of cost estimation tools, especially for atypical projects; how a spiral development process affects the estimation process as compared to the more traditional development approaches; and the need to understand and assess the issues of under-funding and requirements creep on a large, complex project.

**Research Questions**

There is a need to improve the techniques used to develop estimates for software development projects. A methodology for using simulation modeling as a tool for estimating will be obtained and then compared to different models and techniques that are currently used for this purpose in order to answer the following question: What improvements can a cost estimation methodology that utilizes simulation modeling provide?

Other project management areas of interest are the effects of different life cycle processes and the effects of different managerial decisions on a project. Specifically, a spiral development life cycle process will be compared to the waterfall model in order to answer the question: How are cost and schedule affected for a specific project that follows an incremental/spiral approach as opposed to a Waterfall approach?

In addition to the above-mentioned areas of investigation, other key issues for the manager of a large, complex software development project are of interest: What are the impacts of human resource issues such as turnover and experience levels? What are the affects of requirements creep on cost and schedule estimates? These types of questions will be addressed
by utilizing simulation and analyzing the effectiveness of this approach for investigating these questions.

In summary, the following questions will be investigated:

- What improvements can a cost estimation methodology that utilizes simulation modeling provide?
- How can simulation modeling be used to improve cost estimation throughout a project, especially for those that are atypical?
- How does an estimate created with simulation modeling compare to other approaches?
- What benefits are gained by simulation estimation in terms of ability to capture process specifics, midstream adjustments, and adequate tuning of model?
- What effects are seen on cost and schedule estimates when a spiral development life cycle is used instead of the traditional waterfall approach?
- How does the selection of the spiral development lifecycle affect the process of estimation?
- How does the spiral process compare to waterfall process for handling requirements creep for a legacy system replacement project?
- How can the combination of discrete event simulation and system dynamics effectively serve as a quantitative tool for assessing specific managerial decisions in terms of cost and schedule for a large, complex project?
- How is a project affected when it is provided with substantially less funding and schedule than is estimated at the beginning of the project?
- What are the effects of requirements creep on cost and schedule estimates?
In order to meet the above goals of this research, a process model that is based on Raffo’s Process Analysis Tradeoff Method and simulation tool will be used for quantitative and qualitative assessment of a case study project that provides an exceptional opportunity to investigate the research questions of interest. The Process Tradeoff Analysis is a method of combining process models with statistical analysis for assessing the potential impacts of process changes on cost, schedule, and quality. The Process Analysis Tradeoff Tool (PATT) is a process simulation model platform developed by Raffo (Raffo and Wakeland 2003). This tool has already been used to build a process model template for the IEEE 12207 Software Development Process and is intended to assess the impacts of the NASA Independent Verification & Validation (IV&V) activities on the performance measures of cost, schedule, and quality. The model will serve as a baseline model that will be tailored to represent the incremental process utilized by the case study project. This model will then be used to provide cost estimates that can be compared to the project estimates that were created using expert judgment and automated cost estimation tools as well as to the actual project cost data. In addition, system dynamics pieces will be added to the model for analysis of the impacts of other issues such as turnover. The goal is to provide software project managers with a methodology for using a simulation tool that can be tailored to a particular development environment for estimating and for making a quantitative case for deciding among different courses of action, processes, or estimates.

The literature review in Chapter 2 provides an overview of software engineering, discrete and continuous simulation modeling, and the impacts that these models have on large software development projects.
Chapter 3 will outline the methodology to be used in performing this research work. Chapters 4 through 7 will contain findings in the form of pre-press publishable articles. Chapter 8 will provide a summary of the findings and answers to the research questions.
CHAPTER TWO: LITERATURE REVIEW

Software Engineering

The IEEE defines software engineering as “the systematic approach to the development, operation, maintenance, and retirement of software” (Leach 1999). Stated in other terms, software engineering is “a discipline that provides methods to handle the complexity of software development projects with the goal of producing high quality software at low cost” (Jalote 1994). This discipline began in the 1970’s as projects became larger and more complex and there was a recognized need for the same type and level of structure that is found in other engineering sciences. Software engineering utilizes the same problem solving techniques common to all engineering disciplines and provides the basis for project planning, management, design and analysis, validation, and maintenance (Fairley 1985). Software engineering differs from other engineering disciplines because software is intangible and does not have physical properties. The lack of physical laws limits the number of fundamental guidelines and basic constraints available for designing software.

There are several common process lifecycle models for software development. Software models provide guidance on the order in which a project should carry out major tasks. These models attempt to bring order to the complex activities of software development and to help meet the goals of software engineering (Pressman 2001). The waterfall or linear sequential model is the traditional model used for project development (Royce 1970). It consists of a sequential cycle of activities such as requirement analysis, design, coding, testing, and support. It considers requirements to be fixed from the beginning of the project. If requirements are not
well understood at the beginning, this approach is not reasonable. For large complex projects that follow this process, changes to requirements lead to significant cost and schedule impacts.

The widely-used IEEE Standard 12207 for Software Life Cycle Processes (IEEE 1998) is based on the waterfall model and consists of the following activities performed in sequence for the development process: process implementation, system requirements analysis, system architectural design, software detailed design, software coding and testing, software integration, software qualification testing, system integration, software installation, and finally software acceptance support.

Because of familiarity with and heavy reliance on the waterfall model, undesirable planning and management practices have evolved. The sequential nature of the model leads to the belief that each step must be completed before the next one starts. In reality, steps such as requirements development live throughout the development process. Pressure for placing freezes on changes early in the process inhibits creativity and can prejudice measurement as tasks are labeled complete, even when they are not, in order to demonstrate successful accomplishment of the model (Humphrey and Kellner 1989). Due to these issues with the waterfall process, other process models have been developed.

When requirements are known and can be segmented and when it is desirable to have some increments completed quickly, an incremental life cycle model can be appropriate. The incremental process develops a system in increments and emphasizes very short development cycles. This process is also vulnerable to requirements and technical changes since requirements are defined beforehand in order to plan the different increments. Each pre-planned incremental release will add functionality.
Another type of life cycle model is the evolutionary model which takes the incremental model and extends it to the requirements phase (Christensen and Thayer 2001). This model is therefore most appropriate when only general objectives are known or there is evidence that there will be a great deal of requirement volatility. Evolutionary models combine the systematic elements of the waterfall process with the iterative nature of prototyping (Pressman 2001). Prototypes are created for evaluating and refinement of requirements. Prototypes may be used by the customer for a limited purpose, but the main intent is to use them to investigate requirements that need to be fully implemented in the final deliverable. Prototyping is popular in systems with user interfaces and databases and is known as Rapid Application Development. Rapid development is achieved by using component based construction.

The spiral development model is an evolutionary model that was developed by Barry Boehm and which evolved based on experience with large government software projects (Boehm 1988). The goal was to refine the waterfall model in order to better meet the needs of such projects. Boehm defines the spiral model in the following way, “The Spiral Development Model is a risk-driven process model generator for guiding multi-stakeholder concurrent engineering software-intensive systems. Its distinguishing features include a cyclic approach for incrementally growing a system’s degree of definition and implementation, and a set of anchor points milestones for ensuring feasibility of the incremental definitions and implementations” (Boehm 2000). The first increment is a core product with basic requirements addressed, but with supplementary features to be handled with expanded increments of the operational core product. This model emphasizes assessment of risk for each prototype. The development consists of repeating cycles of determining objectives, evaluating alternatives, prototyping, and developing, and then planning the next cycle. Development builds on top of the results of the previous
spirals. Figure 2-1 on the following page is a diagram of the spiral model of the software process that is provided in Boehm’s 1988 article titled, “A Spiral Model of Software Development and Enhancement.”

Figure 1: Spiral Development Model (Boehm 1988)

The radial dimension in this figure represents the cumulative cost incurred in accomplishing the steps to date. The angular dimension represents the progress made in completing each cycle of the spiral. The figure depicts the concept that each cycle involves a progression that addresses the same sequence of steps for each portion of the product. This sequence of steps begins with identification of the objectives of the portion of product being
developed, the alternative methods of implementation, and the constraints such as cost and schedule imposed on the alternatives. The next step is to evaluate the alternatives in relation to the objectives and constraints. Major sources of risk will be identified and cost-effective strategies to deal with these risks should be developed. The government and military are using this process more often in order to overcome the limitations of the more traditional processes. The US Dept. of Defense has determined that the spiral development model is the preferred method/process for software-intensive development lifecycles (Surber 2004). NASA’s new Exploration Program is considering the use of SDM for hardware as well as software development. The Spiral Development Model concentrates on risk management and proper requirements evolution, but problems can arise if unplanned growth is handled by deferred planned functionality. This can quickly get out of hand and lead to an insurmountable level of deferred functionality. When this happens, the process is known as the “Death Spiral” (Brown 2004).

The spiral development lifecycle model can also serve as a process model generator for determining the most appropriate process model based on the process objectives and constraints and major sources of risks (Boehm and Belz 1989; Boehm 2000). The spiral model paradigm clean lead to other lifecycle approaches depending on where the greatest amount of risk exists. For major risk in budget and schedule predictability and control, the spiral process leads to an equivalent waterfall model. For high risk in getting wrong user interface or user support requirements, the spiral becomes the equivalent of an evolutionary process. If the high risk elements of a project involve a mix of risk items, the spiral approach will reflect an appropriate mix of process models (Boehm 1988). Therefore, the spiral model offers a flexible, risk-driven approach to developing software that an often match the reality of a project better than
document-driven models such as the classic waterfall (Tamai and Itou 1993; Hendrix and Schneider 2002). The following critical process drivers can be considered during early spiral cycles:

- **Growth Envelope**: the limits of a system’s requirements especially in terms of size and diversity of function
- **Understanding Requirements**: if requirements are not well understood, use process models that utilize prototyping and evolutionary development
- **Robustness**: If system must be error-free and highly robust, models such as evolutionary pose too much risk because they are informal
- **Available Technology**: Use straightforward application of technology such as Commercial Off The Shelf (COTS) if capabilities can cover growth envelope
- **Architecture Understanding**: waterfall approach poses high risk if system architecture is not well understood

As an example for using these criteria to select a process, a waterfall model is appropriate for rebuilding a legacy system since the growth envelope is large, the understanding of requirements should be high, the robustness is high, and the architecture understanding is high. Incremental development can be used with approaches such as evolutionary or waterfall by organizing the development into a series of increments of functional capability in order to handle any of the following conditions: early capability needed; limited staff or budget available; downstream requirements poorly understood; high-risk system nucleus; large to very large application; or required phasing with system increments (Boehm and Belz 1989).

The more traditional lifecycle models all have certain aspects in common that are crucially different from the spiral model. A key difference is that the more traditional methods
seek to define all requirements at the beginning and are considered as a “do each step once” even though they are meant to be iterative and recursive in their life-cycle applications. (Surber 2004). Spiral development, on the other hand, limits requirements development in each cycle, is risk driven and seeks to hold schedule constant. The thought is that it is better to develop software intensive systems in spirals to resolve inherent uncertainty in getting user confirmation of deployed functional capability on first pass (Boehm 2000). Another key difference is that the Spiral Development Model (SDM) is geared toward the development of new software systems, while the traditional methods have been proven on legacy as well as new systems.

The following are the key sequence of events that should take place in one pass of the spiral:

- **Performance Objectives**: Stakeholders agree and understand that schedule and cost will be held firm.
- **Risk Assessment**: Technical, Cost, and Schedule
- **Design**: Using traditional methods with traceability from requirements to functions to architecture and finally to the design solution
- **Code/Fabrication/Integration**: Design solution with limited technical capability, but gets to end-users quickly
- **Experimental Unit, Test, Tradeoff** – Verify prototype meets requirements
- **Delivery** – Obtain user feedback
- **Lessons Learned** – Review customer experience for next cycle and obtain consensus on goals/targets for next cycle.
Project management can be divided into the general categories of project planning, monitoring and control. Project planning begins before the development activity begins and occurs at a time when there are a lot of unknowns about the project. Therefore, this is a very important and difficult part of project management. A major aspect of the planning activities is developing an estimate for cost and schedule. Estimates that are easily understood and based on a well-founded, verifiable process prepare the project manager for project negotiations and allow for arguments against committing to projects with unrealistic dates and budgets (McManus and Wood-Harper 2003). In an article on software management best practices (Brady and DeMarco 1994), Brady and DeMarco state that, “If we did any other project activity as badly as we set schedules and budgets, the software industry would still be trying to get its first program up and running”. They go on to explain that a failure to deliver a product on time is really a failure in estimating, but failures in performance and productivity are usually given the blame.

Early project estimation based on partial knowledge of a project’s requirements is extremely difficult and the probability that these estimates remain unchanged throughout the life of the project is extremely small. DeMarco states that by default, “An estimate is the most optimistic prediction that has a non-zero probability of coming true” (DeMarco 1982). Brooks highlights the special difficulty of estimating software projects with his statement that “All programmers are optimists” (Brooks 1978) because programmers assume that all will go as planned. In actuality, many things will not go as planned. For instance, resources may not be available as planned, there may be more necessary redesign and rework than planned, and these impact cost and schedule. Even though the initial estimate is the most uncertain, it is in many
ways the most important because initial estimates often become the official estimate for a software project. The important process of developing an initial estimate should lead managers to consider the important factors that will bear on the size and complexity of the effort (McGarry, Waligora et al. 1990). The traditional approach of creating an initial estimate and sticking with it does not take into account the fact that as a project progresses through the software development cycle, it becomes better defined. This also does not take into account the uncertainty that exists for an estimate and the fact that estimate values for time to complete and number of person-months of effort should not be considered point values, but rather ranges with confidence intervals (Putnam 1997). NASA’s Manager’s Handbook for Software Development recommends that a minimum of five re-estimates should be made after the initial estimate at key life cycle phase points. This handbook also shows how uncertainty decreases with an uncertainty proportion that decreases from 1.00 with the initial estimate to 0.05 with the sixth estimate after system test.

The fact that estimating techniques give a probability for project cost and completion data, rather than a specific number or date is often overlooked since managers cannot commit to a probability, but must commit to a specific date or number when beginning a project (Armour 2002). Probabilities are turned into dates via the commitment process that should have the goals of managing risk and optimizing the probability of success. Abdel-Hamid points out that software estimates should have adaptive, corrective, and perfective qualities (Abdel-Hamid 1993). Estimates need to be able to adapt to changing environments; corrective action is needed for remedying faulty initial assumptions such as product size, which is often the case; and lessons learned from completed projects should be done in order to improve project statistics and their usefulness as the basis of future estimates. A project’s estimates are also very important
because they create pressures and beliefs that directly influence the decisions team members make and the actions they take (Abdel-Hamid and Madnick 1991).

The above arguments lead to the conclusion that “software estimation is a continual process that should be used throughout the life cycle of a project.” (Agarwal, Kumar et al. 2001). The procedures that are part of the cost estimation process include: estimate size, estimate cost and effort, estimate schedule, estimate critical resources, assess risks, inspect/approve, track and report estimates, measure and improve the process. The most widely used cost estimation tools such as COCOMO rely on the most critical independent variable of the projected size of a software project and a small error in projected size can lead to a very large error of estimated value of effort. The authors Musilek, Pedrycz, and Succi suggest that management should not be given a single number estimate, but should have the data presented so that its uncertainty is properly expressed so as not to mislead decision makers (Musilek and Pedrycz 2002). Boehm states that humans are often willing to overlook accuracy for precision when it comes to an estimate. People are more comfortable with a “single value that pretends to certainty (and which the estimator “knows” is probably wrong) over a range value that almost certainly includes the most probably value and hence is correct, but which retains some level of uncertainty.” (Boehm and Fairly 2000). Simulation is a tool that can be used to express the uncertainty in an estimate and simulation models can enable continuous updating of estimates by comparing current performance to probable project outcomes. Additionally, they allow for analyzing “what if” scenarios and their effect on estimates better than analytic models.

Measurement is vital to proper estimating and should not be separated from it. The collection and use of metrics is an important part of software engineering in that these activities enable project managers to monitor and control a project. Project control is fundamental and can
affect other project activities such as estimation. Ray Turner (Turner 1984) states that, “Poor project control doesn’t cause bad estimates but it may make good estimates bad”. Software metrics are quantifiable measures of various characteristics of the software itself or the development process (Jalote 1994). Software metrics consist of a variety of activities such as the representation of characteristic properties of software code by numbers, the use of predictive models that provide resource requirements and software quality, and quantitative quality control through monitoring of defects during development and testing (Fenton and Martin 2000). There are more than 500 metrics that have been proposed for the operation and development of software products (Christensen and Thayer 2001). Therefore, selecting a manageable number of useful metrics can be very challenging.

Metrics should reflect which system characteristics are important to managers and to people who execute the process and should also support decisions on the selection of different alternatives (Raffo 1993). The most widely used technique for helping to identify useful metrics for process improvement for an organization is Basili’s Goal Question Metric (G/Q/M) method (El Emam, Moukheiber et al. 1993) (Basili and Rombach 1987). This method provides a framework for stating goals and developing questions about the development process and product that provide a specification for the data needed to help answer the goals. This process of quantifying improvement goals consists of generating a set of goals based upon the needs of the organization, deriving a set of questions or hypotheses which quantify these goals, and developing a set of metrics that provide information needed to answer questions of interest.

The four reasons for measuring software projects are characterization, evaluation, prediction and improvement (Pressman 2001). Characterization allows for improved understanding of the software process, project, or product. Evaluation allows the determination
of status with respect to plans. Prediction allows planning and also allows for updating cost, schedule, and quality estimates based on current data. Characterization, evaluation, and prediction can lead to improvement of the project, process, and product. Software project measurement requires the collection of several different kinds of information in order for the resulting data to be useful. The first kind of information is size information for the deliverables. The second kind of necessary information concerns staffing, effort and cost data for all activities that are part of the project. The third kind deals with defects or bugs since the cost of defect repairs can greatly impact cost (Jones 2000). The fourth kind of information deals with the specific processes and tools used. Process indicators enable an organization to gain insight into the effectiveness of the process and this enables management to assess how things are working.

The most commonly used metrics are the size of the software system, the quality of the system, the system’s performance, and cost (Leach 1999). The most commonly used measurement of the size of the system is the number of lines of code (LOC). Yet, there is a great deal of disagreement on the validity of this measurement. Those who agree with this method argue that Lines of Code is a common artifact that can easily be counted and for which a great deal of literature and data already exists. Opponents argue that LOC measures are programming language dependent and that they do not consider that more than half of all software effort in not directly related to source code (Jones 1998). Argued another way in (Armour 2002), size for estimation purposes should be based on the quantity of knowledge that must be obtained and the difficulty of obtaining such knowledge. However, there is no good way to represent or capture this type of information. In spite of these issues, LOC continues to be the most widely used method of sizing a software project, especially for government and military projects. Capers Jones (Jones 1998) argues that a more accurate method for estimating the size of a project is
through the use of Function Points, an indirect method that attempts to measure the functionality of the software product independent of the language used. It can also be argued that even though the Function Point method counts things such as inputs and outputs, this method still does not fully capture the complexity of a system (Armour 2002).

The problem with accurately sizing a project is recognized as a major problem since size dictates many aspects of a project such as complexity and cost. Product size is a main unknown and growth of a product’s size during development can cause a schedule that was properly based on empirical data to end up wrong (Brady and DeMarco 1994). Experts such as Boehm believe that the software undersizing problem is the most critical road block to accurate software cost estimation (Boehm 1981). A major cause for undersizing is the powerful tendency to focus on the highly visible components. It is also believed by experts such as Boehm that management underestimates a project’s size by factor of 1.5. Size is a critical factor in deriving cost and schedules with models such as COCOMO and undersizing can lead to schedule compression (Sengupta and Abdel-Hamid 1996). Underestimation of project size can lead to another common problem for software development project managers known as the “90% syndrome” (Abdel-Hamid and Madnick 1991). This term describes the phenomenon of a project completing 90% of the scheduled work according to plan, with the final 10% of the work taking more than twice the originally planned schedule. Abdel-Hamid used his continuous simulation model to investigate this syndrome and found that estimating project size and delays in error detection are critical elements in the 90% syndrome. A significant reduction in progress later in a project is typical of those experiencing this syndrome as is the late discovery of unplanned rework. This phenomenon affects all types of projects and it is estimated that the elimination of this syndrome could reduce development cycle time by roughly 50% (Ford and Sterman 1999).
Managerial decisions and process structure have also been shown to contribute to the 90% syndrome.

There are two common measurements of system quality. These are the number of defects or deviations from the specifications and the number of faults per thousand hours of operations. A software fault is some deviation from the requirements of a system and a failure is the inability of the system to perform its essential duties (Leach 1999). Putnam argues that metrics such as these allow for confusion over whether or not the number of defects is defects remaining or defects counted (Putnam 1991). He argues that a better measure is that of mean time to defect because you can more easily relate this to the mission by considering the requirement of how long the software must run continuously without failure.

The metric of choice to gauge where a project is and where it should be has been the number of source lines of code produced per man-month, yet this is now recognized as one of the worst possible metrics. This metric can be wrong as much as 90% of the time because the number of lines and number of man-months both vary with a variety of factors dealing with management practices and the environment (Putnam 1991). A simple set of single-valued metrics that are easy to understand and that do not contain ratios that behave against intuition is the best way to measure progress in a development project. The following five simple metrics have worked well: quantity of function (source lines of code and function points), schedule (the elapsed calendar time), people (the monthly head count), effort (the sum of the people applied over time), and defects (the number of problem trouble reports over some interval of time) (Putnam 1991).

Another metric approach that is used widely for monitoring and controlling the process of a complex project is Earned Value (Fleming and Koppelman 2000; Boehm and Huang 2003)
This is a means of measuring work accomplished by integrating cost, schedule, and technical performance into one set of metrics. With this approach, a set of necessary tasks with associated budgets and schedules are developed and each task is assigned an earned value for its completion, usually the task budget. These values are used to provide feedback on how the project is progressing with respect to the project’s plan and to enable the forecasting of probable final cost and schedule results from as early as the 15% completion point. However, earned value is “cost-oriented” and therefore does not provide information about the other types of value that is provided to the customer or organization by the project (Boehm and Huang 2003).

Norman Fenton, author of a leading text on software metrics, writes that most software metric activities fail to address their most important requirement of providing information that allows for quantitative managerial decision making during the software lifecycle (Fenton and Martin 2000). He also states that the large amount of software metrics activity is not as much the result of companies being convinced of their usefulness but rather is something done only in desperation when there is a problem or when there is a need to satisfy some external reviewer. He goes on to further suggest that the largest reason for metrics in the United States is due to the Software Engineering Institute’s CMMI, since evidence of use of metrics is necessary for achieving higher levels of the CMMI. Defining and using software metrics is important for understanding and improving the software process (El Emam, Moukheiber et al. 1993). The CMMI is the best known model for process improvement based on process assessment.

The idea that improving software processes leads to higher quality software that is developed in a timely manner and at a predictable cost is based on the premise that “The quality of a software system is largely governed by the quality of the process used to develop and maintain it” (Humphrey and Kellner 1989). Because of the engineering aspects of software
development, there can be a tendency to overemphasize methods and tools, although process and people issues are extremely important to producing quality products (Christensen and Thayer 2001). The CMMI emphasizes that maturity of processes is more important in achieving successful software development than the use of advanced technology. There are five capability levels that are defined in the Software Capability Maturity Model. The following list describes the five levels of the Software Capability Maturity Model (Carnegie Mellon University/Software Engineering Institute 1994):

- **Level 1**: Initial – Process is ad hoc and occasionally chaotic. Few processes defined
- **Level 2**: Repeatable – Basic project management process established to track cost, schedule, and functionality
- **Level 3**: Defined – Software process for management and engineering activities is documented, standardized, and integrated into a standard software process for the organization
- **Level 4**: Managed – Detailed measures of the process and product quality are collected and both process and products are quantitatively understood and controlled
- **Level 5**: Optimizing – Continuous process improvement is enabled by quantitative feedback from the process

Identifying appropriate metrics is fundamental to understanding the software process and improving it because software metrics may be used for assessment or prediction. The combination of metrics and predictive models provides more insight into the project than what is obtainable through the use of metrics alone and can provide information to managers when they consider process tradeoffs in attempting to bring a project back on track (Raffo, Harrison et al. 2002).
Models

Process simulation models, system dynamics models, and static cost models already exist for software development projects. Each of these tools has advantages and disadvantages and the appropriateness of each depends on the application. The models found in the literature are based on waterfall-type software development life cycle processes. Cost estimation models, also known as analytical summary models, assume static values for the product, personnel and environment, but do not explain why or under what conditions the mathematical relationships between variables are applicable. The main advantages of this type of model are that they contain objective and repeatable formulas; they are based on empirical data; and they can be used to predict the impact on dependent model variables given changes in high-level model inputs. Some disadvantages are that they do not capture process specific issues; they are not well suited for midstream adjustments after a project starts; and they do not model the dynamic aspects of a project that can affect productivity such as managerial decisions.

System dynamics models provide a coarse representation of the process and provide an integrative model that includes interactions between related project activities. This type of model contains interrelations and dependencies of software development at a more in-depth level than analytic summary models. Advantages of this type of model are that they allow for analysis of the effects of select relationships and interdependencies that are described by objective, repeatable differential equations, and that they can capture the dynamic interrelationships between technical and social factors. The main disadvantages of this type of model are that they can contain subjective inputs, they do not capture process specifics, and they treat tasks in the “aggregate” as opposed to providing detail on individual tasks. These models, like all models,
need to be modified to handle different interdependencies and new key factors due to changing environments.

Process models contain more of the intricacies of the specific process being used or investigated than system dynamics models. This type of model is the only type described that can provide operational guidance and the ability to check the integrity of the process. Other advantages are that objective, repeatable formulas are used and these models are able to handle exceptional circumstances. Disadvantages are that the focus of a particular model is on a single process and therefore, results may not be applicable to all situations, subjective inputs are possible, and the model may overlook system level costs. As with system dynamics models, modifications must be made to the model in order to handle changes or alternatives that were not foreseen or explicitly represented.

The most widely used and recognized cost model is COCOMO (Constructive Cost Model) (Boehm 1981). This analytical model was proposed by Barry Boehm in 1981 and is used for software labor and schedule estimation. The fundamental equation for COCOMO is the Effort Equation that is used to estimate the number of Person-Months required. Three different models are available: Basic, Intermediate, and Detailed. More factors are included in the more complex models to allow for creating more accurate estimates. Popular software estimation models such as COCOMO are not well suited for an iterative software process since a conventional process experience base is used (Royce 1998).

An updated and recalibrated version of COCOMO known as COCOMO II (Boehm, Abts et al. 2000) was released to reflect changes in software development practice such as the change from overnight batch processing on mainframes to real-time on desktops; an increased emphasis on reusing existing software and using off-the-shelf components; and emphasis on the
development process as much as on the product (Boehm, Abts et al. 2000; USC 2004). In order to avoid confusion, the nomenclature for the original COCOMO model published in 1981 became known as COCOMO 81. COCOMO II consists of three sub-models: Application Composition, Early Design, and Post-Architecture. The Application Composition model is used for projects using Integrated Computer Aided Software Engineering tools for rapid application development. Early estimate is used for rough estimates when there is incomplete project information. Only the last and most detailed model, Post Architecture, has been implemented in a calibrated software tool. COCOMO II added the following new cost drivers: application precededness, development flexibility, architecture and risk resolution, team cohesion, process maturity, required software reuse, documentation match to lifestyle needs, personnel continuity, and multi site development. Eliminated were COCOMO 81’s concept of development modes and turnaround time and modern programming practices cost drivers (Clark and Devnani-Chulani 1998). The cost drivers in COCOMO II must be assigned values which are then used to estimate the effort, cost, and schedule. The literature emphasizes the fact that it is not always possible to obtain a high level of confidence in these factors due to the ever-changing software development environment and the reliance on subjective expert evaluations for the values (Musilek and Pedrycz 2002; Tian and Noore 2003). In addition, since these factors are not weighted equally and since they may not all be totally independent, quantification of their relationship is extremely difficult.

Analytical models such as COCOMO use mathematical relationships between variables and provide steady-state values that result from changing the values of the variables. These models do provide for limited “what if” scenario exploration by demonstrating the effect of adjusting requirements, resources, and staffing might have on predicted costs and schedules.
Computer-based simulation models, on the other hand, can specify a system’s continuous transitions and intermediate states and allow for a much wider range of “what if” scenario investigations by evaluating different assumptions and factors, such as process, managerial, and behavioral factors, and their impact on project performance. (Menzies and Sinsel 2000) demonstrate how an analytical tool such as COCOMO II is limited in its ability to handle incomplete information, uncertainty, and “what if” investigation. They show how the number of COCOMO II runs that would be necessary to fully assess the outcome performance measures for all combinations of current and possible values gets prohibitive extremely quickly. The authors show how simulation can be used to select smaller, more significant ranges of the factors that should be tested to make the testing more practical and the results more meaningful. “What if” investigations with analytical tools are also limited by the cost drivers since they are the only factors that can be adjusted. Simulation, on the other hand, can be used to model and test more specific process and behavioral scenarios and is not limited by a set of predetermined factors.

**Computer Simulation**

Simulation (Law and Kelton 2000; Kelton and Sadowski 2004) is the method of designing a computerized model of a system for the purpose of conducting numerical experiments. Computer simulation uses software that is designed to mimic a system’s characteristics over time in order to gain understanding of the system’s behavior for certain given conditions. Simulation is very useful for studying complex systems and dynamic behavior in social systems. Computer simulation tools can serve as a laboratory tool for testing ideas and
hypothesis. This is especially desirable for software engineering due to the difficulty of testing hypotheses (Abdel-Hamid and Madnick 1989).

In continuous simulation, the state of the system can change constantly over time. In discrete simulation, changes occur only at separated points in time. Many systems must be modeled with some random input components and are therefore stochastic in nature. Stochastic simulation models produce random outputs that only estimate the true characteristics of the model. This is one of the main disadvantages of simulation (Law and Kelton 2000).

Simulation models have three main benefits over analytical models (Abdel-Hamid 1993). The causal structure provided by computer simulation allows for explanatory power that is not present with analytical models. For example, regression-based models are unable to provide an understanding of why a statistical relationship exists between development factors and effort. Another benefit is that computer-based simulation models depict a system’s continuous transitions and this is not possible with analytical models. They can only specify the steady state result from changing values of the controlled variables. The final benefit of computer simulation over analytical methods is that it allows for testing of the effects of different assumptions and environmental factors. This is important for complex systems since this will offer knowledge of interactions that would not be possible through investigation of the real system.

**Software System Dynamics**

System dynamics refers to the simulation technique developed by Forrester (Forrester 1961; Forrester 1971) that is “a computer-based replica of organizational reality” (Abdel-Hamid 1996). Models are formulated using continuous quantities that are expressed as levels, rates and
information links that represent the feedback loops and are comprised of coupled, nonlinear first-order differential equations. System dynamics is a simulation technique that can create integrative models for software engineering to analyze interactions between activities that are related such as testing, hiring, training, etc. (Sterman 2000). System dynamics modeling is also recognized as a useful tool for conducting research in dynamic decision making such as how software project managers handle staffing delays and how their decisions affect the outcome of a project or how the presence of unreliable initial estimates lead to self-fulfilling prophesies (Abdel-Hamid and Sengupta 1993; Sengupta and Abdel-Hamid 1996; Sengupta and Abdel-Hamid 1999).

Typical problems in software development include poor planning, lack of risk identification and mitigation, constantly changing requirements, and various managerial problems such as poor hiring and training practices. These problems are related and an understanding of their interdependencies can provide information for making improvements (Madachy 1994). An integrative view of management-type functions such as planning, controlling, and staffing and production-type functions such as designing, coding, reviewing, and testing is useful and can identify multiple factors that may be interacting to cause problems. Brooks states in his well-known book, The Mythical Man-Month, that “…No one thing seems to cause the difficulty…..But the accumulation of simultaneous and interesting factors…” (Brooks 1978) There are hundreds of variables that can affect software development and these variables are often related to one another.

System dynamics models utilize dynamic feedback principles to refine and shed light on the intricate network of interacting variables (Abdel-Hamid 1996). One of the most widely known system dynamics model for software development was created by Abdel-Hamid and
Madnick and is supplied in their book, *Software Project Dynamics: An Integrated Approach*. This model helps fully assess the second and third order consequences of management policies and procedures. This high-level model simulates the typical waterfall process after requirements have been obtained, is intended for medium-sized projects, and is used to predict the impacts of various managerial policies. Abdel-Hamid uses the model to test Brooks Law which states, "Adding manpower to a late software project makes it later" (Brooks 1978) and finds that this heuristic is correct, especially in the late testing phases due to increased training and overhead costs (Abdel-Hamid and Madnick 1991). In follow-on work in 1993, Abdel-Hamid created a model that combined his software development system dynamics simulator with the algorithmic estimator, COCOMO. In this work (Abdel-Hamid 1993), he demonstrates how the model can be used before a project begins to adapt COCOMO estimates to reflect the true nature of the staffing limitations, during development to adapt product-sizing assumptions, and after completion to revise and improve estimates through post-mortem analysis.

Abdel-Hamid’s models do not concentrate on different process models and architectures and this is needed since the waterfall process does not always lead to successful performance. Details of the actual development process are missing from the model and it uses only software development rate and allocated staff to determine changes to the completed task level. It is important to examine the relationship between development phases. The model also misses important interactions such as effectiveness of error reduction techniques. This work was an important first step, but more detail is needed for managerial decision-making and planning purposes.

Madachy expands on the work done by Abdel-Hamid in his 1994 dissertation titled, “A Software Project Dynamics Model for Process Cost, Schedule and Risk Assessment” (Madachy
Madachy points out some of the above-mentioned shortcomings of Abdel-Hamid’s model. Madachy’s system dynamics model expanded the Abdel-Hamid model to include major software phases and an inspection-based process. This was integrated with a knowledge-based method for risk assessment and cost estimation. Madachy was interested in answering the following type of questions: What amount of effort should be committed to error removal? What rates should documents and code be inspected for optimal error removal? Madachy argues that discrete event based simulation is not capable of capturing the time-varying interrelationships between technical and social factors as is possible with system dynamics. Managers are often interested in the “big picture” rather than in individual tasks and therefore, tasks can be expressed by a continuous simulation model. In addition to Madachy’s model’s lack of process details, there is no mechanism for queuing tasks before each development phase. Therefore, this had to be handled by allocating staff only at specific times in order to control the level of activity in each phase.

**Process Models**

Software life cycle processes affect the quality, cost, schedule, and responsiveness of the software system. Traditionally, life cycle models have been considered process models. However, these high-level models do not contain enough detail on the actual process steps to provide guidance on how to integrate the numerous process steps that need to be performed. The field of software process modeling (Humphrey and Kellner 1989; Humphrey, Kitson et al. 1989; Kellner 1990; Curtis and Kellner 1992) began with the following objectives: increase understanding of processes; enable processes to be formally defined and applied; support
improvements to a process; and assist management of a project. Much of the work in this area was done at the Software Engineering Institute using STATEMATE, a commercially available system that provides highly visual representation (Kellner 1989). Three types of diagrams are used to represent the functional, behavioral, and implementation perspectives. Connections between the diagrams allow specification of when and by whom a task is performed. Kellner’s initial work concentrated on descriptive models, but the need to evolve models to allow for complete analysis of the process and for predictions regarding the consequences of potential process changes was recognized (Humphrey and Kellner 1989; Kellner 1990). Kellner demonstrated how process simulation models can be used to support project management planning and control (Kellner 1991A; Kellner 1991B) and how the quantitative nature of these models offers advantages over traditional project management approaches such as critical path and PERT. Process models are better able to provide insight into behavior and to illustrate the importance of feedback loops in software processes. They are more general than the traditional methods and lend themselves to resource constraints and full Monte Carlo simulation analysis.

Raffo extended process modeling work to include the assessment of the impact of potential process changes and to incorporate a defect removal model so that quality could be another basis of analysis in addition to cost and schedule (Raffo 1996). He developed the Process Tradeoff Analysis which combined process models with supporting statistical analysis. This methodology led to the development of the Process Analysis Tradeoff Tool (PATT) which is presently being used to assess the benefits of NASA’s Independent Validation and Verification process on software development projects (Raffo and Wakeland 2003; Raffo 2004). The strength of simulation process modeling for capturing specific details of the process has been demonstrated through such work.
Therefore, in summary, system dynamics models utilize continuous simulation and are often better for obtaining information about the behavior of a project under different management decisions while process models primarily use discrete event simulation and are better for answering questions about the effects of particular process steps on cost, schedule, and quality (Martin and Raffo 1997). Thus, in order to capture a wider range of issues and concerns, a combination of simulation modeling approaches may be useful, although this has not been done often. Considering the strengths of each, a continuous simulation can be used to model the dynamic environment while discrete simulation can be used to model tasks and resources (Martin and Raffo 2000). Two different approaches to combining model types are found in the literature. Martin and Raffo (Martin and Raffo 2001) developed a hybrid discrete simulation model with embedded continuous simulation pieces to simulate the effects of resource constraints on parallel activities. Donzelli and Iazeolla (Donzelli and Iazeolla 2001) developed a two-level model that combines analytical, system dynamics, and process modeling to study the effects of requirements instability on the waterfall process. These two examples demonstrate the effectiveness of combining techniques.

Ruiz, et. al (Ruiz, Ramos et al. 2001) point out that the size, complexity, and detail of a model are interrelated and that there are tradeoffs between the power of the model due to much detail and the ease of use of a model. Madachy demonstrates that small models an be highly valuable for providing insight into dynamic trends and states that smaller is better when presenting results to people not familiar with process modeling or simulation (Madachy and Tarbet 2000). Therefore, a model that is powerful due to the numerous possibilities of experimentation and which is also easy to use and understand is desirable and challenging.
Figure 2 summarizes the work that has been accomplished in software modeling to date. Additionally, the intent of this research is presented so that the comparison to and extension of previous work is evident.
<table>
<thead>
<tr>
<th>Author</th>
<th>Boehm</th>
<th>Abdel-Hamid,Madachy</th>
<th>Kellner, Raffo</th>
<th>Martin</th>
<th>Donzelli, Iazeolla</th>
<th>Mizell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Type</td>
<td>Static Cost</td>
<td>System Dynamics</td>
<td>Discrete Event Process</td>
<td>Combination</td>
<td>Combination</td>
<td>Combination</td>
</tr>
<tr>
<td>Characteristics</td>
<td>Effort and schedule equations; Steady-State Values</td>
<td>Waterfall; Effects of relationships, interdependencies</td>
<td>Waterfall; Consequences of process changes</td>
<td>Waterfall; Effects of resource constraints</td>
<td>Waterfall; Effects of requirements instability</td>
<td>Spiral; Estimation; Effects of Uncertainty, Turnover</td>
</tr>
</tbody>
</table>

Figure 2: Summary of Software Development Modeling Work
CHAPTER THREE: METHODOLOGY

The goal of this research is to develop a methodology for using simulation modeling for developing initial estimates as well as updating throughout a project. In general, software simulation models are used for analyzing impacts of process changes or to assess consequences of management policies and procedures. Existing models are based on the traditional waterfall lifecycle. The new application of utilizing simulation models for cost estimation will be compared to other methods of creating cost and schedule estimates. The Process Analysis Tradeoff Tool (PATT) is a process simulation model that was developed to assess cost, schedule, and quality impacts of potential process changes and will serve as a baseline model for this research. PATT presently model’s a traditional development process and does not contain system dynamics pieces. This combination will provide for a more comprehensive tool that is geared toward improved analysis of estimates.

The literature points out the need for simulation models that capture software development life cycle processes other than the waterfall process (Madachy 1994; Donzelli and Iazeolla 2001). Therefore, a new life cycle development process, spiral development, will be created using the PATT platform. The effects of using a spiral development life cycle model as opposed to the more traditional approaches are of interest as this newer life cycle model gains increasing popularity, especially for government and military software development projects. The new model of the spiral development process will be used to investigate the impact of this life cycle model over the traditional waterfall approach. In addition, the inclusion of system dynamics pieces will allow for investigation of specific managerial decisions related to initial estimates and their effect on a project.
In order to meet the goals of this research, a process model will be developed that is based on the Process Analysis Tradeoff Tool, PATT, which is a process simulation model platform developed by Raffo (Raffo and Wakeland 2003; Raffo 2004). PATT has already been used to build a process model template for the IEEE 12207 Software Development Process and is intended to assess the impacts of the NASA IV&V activities on the performance measures of cost, schedule, and quality. This IEEE process and model template will serve as a baseline for developing a model of the spiral development lifecycle process that will then be tailored to a very large NASA software development project that was cancelled prior to completion. The model will be designed for project managers to use as a quantitative tool for investigating software development project management areas of interest or concern.

This research will expand on work that has been done in system dynamics and process simulation modeling by combining the two for the purpose of developing a simulation methodology for estimating throughout a project. Additionally, the methodology will allow assessment of the impacts of managerial decisions and the spiral development process for a specific development environment. Work has been done by Abdel-Hamid (Abdel-Hamid 1993) to investigate how a COCOMO-derived estimate is affected with staffing limitations using a high-level system dynamics model for medium-sized projects. Abdel-Hamid’s work served as a starting point and he recognized the need for significant enhancements to the model in order to be applicable to large, complex projects (Abdel-Hamid and Madnick 1991). Madachy expanded on Abdel-Hamid’s work by modeling an Inspection Based Process using system dynamics to investigate the effects of performing inspections on effort, schedule, and quality for medium to large sized projects (Madachy 1994). System dynamics models are useful tools for demonstrating the dynamic behavior of a project and are based on aggregation of variables and
tasks with no process details or intricacies being captured. Process models, on the other hand, do provide great detail on the process and can be used to provide guidance on the sequence of process steps and information flows and can also be used to analyze proposed process changes. In addition, process models can support management planning and control activities. This type of modeling, however, does not capture the interactions and structural relationships as effectively as system dynamics modeling. Therefore, a combination of both types of models can lead to a more effective tool. Raffo has developed a process modeling platform that presently models a waterfall type process and enables assessment of process changes or enhancements such as Independent Validation and Verification. This research will enhance and expand the use of Raffo’s process modeling tool for portraying uncertainty in initial estimates and providing continuous estimating capability.

The large, complex NASA case study will be used to demonstrate the effectiveness of the model for continuous estimation, predicting the impacts of a spiral life cycle process, and investigation of specific managerial decisions. This case study approach will be used to answer the following questions: How does a cost estimation methodology that utilizes simulation benefit a large, complex project? How is a project affected when it is provided with less funding and schedule than is estimated at the beginning? What are the affects of requirements creep on cost and schedule estimates? What effects are seen on cost and schedule estimates when a spiral development life cycle is used instead of the traditional waterfall approach? The single case study will be used to seek the answers to these questions using simulation modeling. The nature of the project and the large amount of available data provide a unique opportunity to investigate improvements to cost and schedule estimation techniques and the effects of certain managerial decisions using simulation modeling. According to Yin, (Yin 1989), because of the large
amount of available data, a unique opportunity to investigate improvements to cost and schedule estimation techniques and the effects of certain managerial decisions using simulation modeling provide valid rationale for performing a single case study. The NASA case study project has been cited as a classic “death march” project, meaning that it was plagued with budget and schedule overruns and requirements creep (Yourdon 1997). Even though the case study project was cancelled prior to completion, several increments were completed and will serve as multiple units of analysis.

**Process Model Enhancements**

Presently, the PATT process simulation model has been verified using commercial data for a traditional waterfall process, the IEEE 12207 process. This will serve as a starting point for developing a cost estimation methodology that utilizes simulation modeling. PATT consists of six sequential project phases and system integration and planning along with a layer of Independent Verification and Validation activities.
Each of these phases has associated steps which represent the sequence of tasks that are involved in executing the software process. Equations for each process step calculate activity effort, remaining effort, duration, and defect injection rates. The model requests user-supplied data for parameters such as: size (KSLOC), productivity (LOC/HR), defects (per KSLOC), and project size variation (%of total size). The model assumes that a number of components that are of equal size and effort are being developed. A task is defined as a process step or activity performed on a component or item. Each activity in the process is assigned a desired number of staff and an earned value. The earned value represents the percentage of the total effort for the project that is allocated for a given process step for all components. The model presently uses industry averages for project size, productivity, and defects.

The use of probability distributions for key variables that cannot be known with certainty and that have a major impact on cost and schedule will be used in the model. Theoretical
distributions will be used when possible and based on literature data for unknowns such as size. Productivity and defect data from the Software Engineering Laboratory will be analyzed for distributions in order to best capture the specific environment of the case study project. The Software Engineering Laboratory was a joint activity between NASA’s Goddard Space Flight Center, the University of Maryland, and Computer Sciences Corporation that existed for 25 years between 1976 and 2001. Data was collected on over 140 projects and included the projects’ software characteristics (overall and for components), changes and errors during all phases of development, and effort and computer resources used. The stochastic nature of the model will produce staff or duration estimates that demonstrate the level of uncertainty for estimates. As data is received for the first increment, parameters may be adjusted to reflect what was happening early in the project. The project’s staff/resources will be continuously adjusted as managerial decisions greatly impact the human resource portion of a project and its ultimate success. Boehm states that “after product size, people factors have the strongest influence in determining the amount of effort required to develop a software product” (Boehm, Abts et al. 2000). The interrelationships of personnel factors and their dynamic effects will be assessed by combining system dynamics output with the discrete model.

The effects of under-funding and requirements creep will be assessed considering turnover rates and staffing experience levels. These factors and their interrelationships impact a project. For example, the turnover rate of project employees is a key management indicator as low attrition of good staff is a sign of success. The industry average is 30% and an increase in unplanned attrition is a strong indicator that a project may be headed for trouble (Royce 1998).

These initial changes to the baseline PATT will serve as the engineering portion of the incremental spiral model to be developed. See Figure 4 below. Additional blocks will be added
to cover risk assessment, demonstration, and lessons learned and the entire process will be repeated for a selected number of increments.

Figure 4: Spiral Development Model

The model will be verified after each major change to the baseline model to ensure that it runs properly with no hang-ups and to verify that reasonable results are obtained. The model will be validated to ensure that the model is an accurate representation of the case study project that is being analyzed. This will be accomplished by modeling the lifecycle process that was used on the project and by obtaining feedback on face validity from project personnel. Distributions will be fitted to historical Software Engineering Laboratory data. This organization is well respected by the NASA software development environment and should provide for confidence in the data. Finally, actual project data will be used in the model to see if project behavior can be reasonably predicted by the model.
A flowchart of the approach that will be taken to accomplish this work appears in Figure 5 and is followed by a brief description of each activity.

1. a) Collect and analyze NASA project data. Data will be collected for “snapshot” timeframes including beginning of project and each increment. Data used for various estimating methods.
as well as high-level productivity metrics such as time, effort, size, complexity, monthly and yearly expenditures and accomplishments, and defect profiles will be collected.

b) Develop key theoretical distributions based on Software Engineering Laboratory productivity and defect data

2. Add system dynamics pieces to represent changing human resource environment in model.

3. Develop incremental spiral process using PATT as baseline for engineering portion of spiral.

4. Compare simulation estimates with actual estimating methods used.

5. Use tailored PATT model for comparison with initial case study estimate. Use incremental spiral model for comparison with 60 day pilot estimate.

6. Analyze effect of under-funding and requirements creep on case study.

7. Develop methodology for analyzing estimates using simulation
**Model Inputs**

As already mentioned, data from the Software Engineering laboratory and the literature were used to determine stochastic input values for the model.

The uncertainty in the estimated size of the project will be based on Boehm’s size variation according to the lifecycle phase in which the size is estimated. The consideration of uncertainty in sizing a project is useful since determining size is very challenging and the most significant input to a tool like COCOMO is size (Boehm, Abts et al. 2000). The figure below is found in *Software Cost Estimation with COCOMO II* and is used to highlight the great deal of uncertainty that exists when predicting the size of a project, especially very early in a project.
The literature points out the ever-present problem of underestimating the size of a project (Boehm 1981; SEL 1990). Therefore, the following adaptation of Figure 6 will be used as the guideline for the size input to the model:

Figure 6: Software costing and sizing accuracy versus phase

In order to capture the impact and uncertainty of sizing a software development project, a uniform distribution is used for the estimated size of the product. The parameters of the distribution will be based on which phase of the lifecycle the estimate is developed. For example, if a project’s size is estimated to be 1 million lines of code prior to gathering requirements or developing a concept of operations, the input probability distribution for size will be Uniform (1 Million, 4 Million) LOC. An estimate of 1 Million LOC after requirements are documented would equate to a Uniform (1 Million, 1.5 Million) LOC distribution for size.
The Software Engineering Laboratory (SEL) began collecting data for NASA flight software development projects at Goddard Space Flight Center in 1976 and served as a major resource in software process improvement activities. (Basili, McGarry et al. 2002). Extensive project and product data was collected for over 200 projects and is available to the public (SEL 1997). For the purposes of this research, subsets of the data that were collected and organized by the NSF Center for Empirically Based Software Engineering (CeBASE 2005) and were used. Probability distributions were fitted for productivity data and defect injection data using ExpertFit software to determine which theoretical probability distribution best represented the data set. In all cases, the Kolmogorov-Smirnov test was passed with $\alpha = 0.05$.

The productivity dataset was obtained by the CeBASE database that contained overall product information. Productivity was calculated for each project by dividing the total number of lines of code by the total management and service hours. The following presents a summary of the productivity data:

Table 1: Sel Productivity Data (LOC/Hr) Summary

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of observations</td>
<td>142</td>
</tr>
<tr>
<td>Minimum observation</td>
<td>0.28</td>
</tr>
<tr>
<td>Maximum observation</td>
<td>10.3</td>
</tr>
<tr>
<td>Mean</td>
<td>4.1</td>
</tr>
</tbody>
</table>

The probability distribution of Erlang $(1.36, 3)$ was selected for productivity. Erlang distributions are frequently used for queueing theory to represent service times.
Analysis of project defect data was more complicated due to the difficulty of correlating defects per week to specific projects so that defect densities (defects per KSLOC) could be calculated for each project’s lifecycle phases. Therefore, a smaller subset of data that consisted of correlated defect data from 10 projects was used. This data is available from The Web Measurement Environment at http://asgard.fc-md.umd.edu/webme.

Defects per week were divided by lines of code added per week for each of the projects. The weekly defect density data was then separated by the phases of requirements, design, coding, and testing during which the defects were identified. The following two tables summarize the defect data:

Table 2: SEL Defect Data (Defects/KSLOC) Summary

<table>
<thead>
<tr>
<th></th>
<th>Requirements</th>
<th>Design</th>
<th>Code</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of observations</td>
<td>37</td>
<td>218</td>
<td>160</td>
<td>79</td>
</tr>
<tr>
<td>Minimum observation</td>
<td>0.03</td>
<td>0.06</td>
<td>0.24</td>
<td>1.3</td>
</tr>
<tr>
<td>Maximum observation</td>
<td>22</td>
<td>138</td>
<td>471</td>
<td>158</td>
</tr>
<tr>
<td>Mean</td>
<td>2.7</td>
<td>13</td>
<td>32</td>
<td>41</td>
</tr>
</tbody>
</table>
Data on defects related to bad fixes and documentation were not available, so a nominal value of 30 defects/KSLOC that is based on the literature (CeBASE 2004) was used in the model for these types of defects.

Lognormal distributions are often used to represent the time to perform an activity or the time between failures. Previous software process modeling work has demonstrated how lognormal distributions can be used in simulation process models (Raffo 1996). Weibull distributions are commonly used for product lifecycles and reliability issues. Exponential distributions are the most common distribution of choice for business processes and often are used to represent activity times.

In the event that the Software Engineering Laboratory data is not appropriate for a certain development environment, there are two options to determine appropriate theoretical probability distributions for productivity and defects. The first option is to gather and analyze historical data from similar projects or environments. This is not always an easy task as adequate data collection for large software development projects is costly and time-consuming. If adequate historical data is present, then a program such as ExpertFit can be used to determine the best theoretical fit, if one exists. If adequate historical data is not available, then it may be necessary to estimate parameters for easy to use and understand probability distributions. For the productivity distribution, management can decide on suitable values for a triangular distribution.
by estimating the maximum, minimum and most likely productivity values. For defect data, estimates for reasonable mean and standard deviation values can be used to determine appropriate lognormal distributions.

**Model Output**

A goal of this research is to determine the benefits of using simulation for cost estimation for large, complex software development projects. Therefore, values for total effort and project duration are the output parameters of interest.

The following data is displayed on the main screen of the model for each run set:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td></td>
<td></td>
<td>KSLOC</td>
</tr>
<tr>
<td>Effort</td>
<td></td>
<td></td>
<td>Person-Months</td>
</tr>
<tr>
<td>Rework Effort</td>
<td></td>
<td></td>
<td>Person-Months</td>
</tr>
<tr>
<td>Duration</td>
<td></td>
<td></td>
<td>Months</td>
</tr>
<tr>
<td>Average Duration</td>
<td></td>
<td></td>
<td>Months</td>
</tr>
<tr>
<td>Injected Defects</td>
<td></td>
<td></td>
<td>Defects</td>
</tr>
<tr>
<td>Detected Defects</td>
<td></td>
<td></td>
<td>Defects</td>
</tr>
<tr>
<td>Corrected Defects</td>
<td></td>
<td></td>
<td>Defects</td>
</tr>
<tr>
<td>Latent Defects</td>
<td></td>
<td></td>
<td>Defects</td>
</tr>
</tbody>
</table>

Figure 8: Process Model Main Screen Output Display
More detailed data on parameter values for each replication can be found in the output database. The total effort is the addition of effort and rework effort and this parameter is found in the output database as well. This data can be used to calculate confidence intervals for the two primary estimation parameters of effort and duration. The confidence intervals for total effort and duration will be key values for very early estimates and can serve as a check for early estimates developed with other tools and techniques as well as a check on the validity of predetermined budgets.

Use the following equation for confidence intervals (Kelton and Sadowski 2004):

$$\bar{X} \pm t_{n-1,\frac{\alpha}{2}} \frac{S}{\sqrt{n}}$$

Equation 1: Confidence Interval Calculation

Where:

- $$\bar{X}_{(n)}$$ = the mean of the sample of size n
- $$t_{n-1,\frac{\alpha}{2}}$$ = the Student t value for n-1, 1-$$\alpha$/2
- $$\alpha$$ = the desired level of confidence
- $$S^2$$ = the sample standard deviation

Running more replications will reduce the half-width of the confidence interval because as the sample size increases, the variability of the sample mean decreases. The following approximation can be used to decide how many replications are needed if it is assumed that there will be more than 30 replications (Kelton and Sadowski 2004):
\[ n \geq n_0 \frac{h_0^2}{h^2} \quad \text{Equation 2: Number of Replications Calculation} \]

Where

\[ n_0 = \text{number of initial replications} \]
\[ h_0 = \text{initial half-width} \]
\[ h = \text{desired half width} \]

Start with an initial run set of five and use the results to determine how many replications should actually be run in order to reduce the half-width to less than 5%

**Addition of System Dynamics**

The discrete event process model utilizes four resource pools: development staff, IV&V staff, Quality Assurance (QA) staff, and Other staff. The user supplies the number of staff in each pool and the model draws resources from the appropriate pool according to the task that is being performed. The model does not distinguish between experienced or inexperienced staff. An area of interest of this research is to demonstrate the usefulness of combining system dynamics with discrete event process modeling in order to benefit from the advantages of both types of models. Specifically, the effect of experience levels and turnover on productivity and then ultimately on cost and schedule is of interest. Therefore, a system dynamics program will be used to consider the effect of turnover on staff experience levels. In the system dynamics model, resources are divided into experienced and inexperienced groups and the time-changing levels will be derived based on a turnover rate and assimilation rate. The ratio of experienced
staff to total staff will be calculated and then sent to the discrete event model to affect productivity.

The System Dynamics model was created in Vensim software. Vensim models graphically display the connections and feedback loops of the system. It is possible to instantly see simulation results for all variables on the screen and it is possible to view more detailed results of any selected variable of interest with different analysis tools. It is very easy to perform simulation tasks and the powerful SyntheSim mode provides attached sliders for each model constant that can be used to adjust values and to instantly observe the effects of the adjustments. For each variable there is either a superimposed graph or a slider below. Sliders are for constants that can be easily changed and thumbnail time graphs are shown for the remaining variables. As you move a slider, the new results will be displayed in blue, with the baseline results displayed in red for all variable graphs. The graphs will update instantly as constant sliders are moved and the model is simulated.

Figure 9 shows the Vensim diagram of a simplified Human Resource system dynamics model that is based on the work of Abdel-Hamid and Madnick (Abdel-Hamid and Madnick 1991).
The desired output from the system dynamics model is the number of experienced personnel available throughout the project. This number will be used to develop a ratio multiplier to productivity and will be used in the discrete event software process model. It is assumed that an inexperienced person is 50% as productive as an experienced person. Since a ratio of 1 for \( \frac{\# \text{experienced staff}}{\# \text{total staff}} \) would equate to a productivity multiplier of 1 and a ratio of 0 for this quantity would equate to a multiplier of 0.5, the following equation will be used to calculate the productivity multiplier:

\[
\text{productivity multiplier} = 0.5 \left( \frac{\# \text{experienced staff}}{\# \text{total staff}} \right) + 0.5
\]

Equation 3: Productivity Multiplier Calculation
The system dynamics model will be used to calculate the productivity multiplier for each day during the project and this data will be read into the discrete event model. Each time the discrete event model attempts to draw a productivity number from the productivity distribution, the time will be captured and the associated productivity multiplier will be used in the calculation to affect the productivity draw.

After each simulation, the table output of number of experienced staff for each day will be exported and saved as a text file. A simple Perl routine is used to calculate the productivity multiplier and to store the data in a single column text file where the row numbers correspond to the days of the simulation. A Global Array Load block was then added to the discrete event process model. This block will load data from the productivity multiplier text file and will save it in an array that is indexed by days. Each time the discrete event process model draws a productivity number from the productivity distribution, the discrete model’s current simulation used to draw the corresponding multiplier from the array so that it can be multiplied to the productivity. Below is a flowchart of this process.

Figure 10: Flowchart of Process to Combine Models

In order to test the combination model, two extreme data sets were used. The first represented the case where all personnel were experienced and there is no turnover. For this scenario the productivity multiplier will be 1 for every day in the simulation. This data was imported into the model through the global array and the resulting effort and duration exactly
matched that of another run set of the discrete event process model that did not contain the global array load block. The second extreme case used as a test was for an all inexperienced workforce that will remain inexperienced throughout the course of the project. For this scenario, the productivity multiplier will be 0.5 for each day in the simulation. The effort and duration were more than double that for the baseline run set of the model.

**Incremental Spiral Development Model**

The definition of spiral development as a cyclic approach for incrementally growing a system’s degree of definition and implementation was used as the basis for development of the incremental process development model. Repeated loops of the waterfall process are used with the added activities of risk assessment and lessons learned. Figure 4 shows how the PATT IEEE 12207 process model serves as the core of the incremental model.

![Figure 11: Use of PATT for Spiral Model](image-url)
Activity blocks for risk assessment and lessons learned were created and added to PATT to expend effort and schedule. The following table is based on the literature (Boehm, Abts et al. 2000; Hendrix and Schneider 2002) and provides values for the % of overall effort spent in each spiral activity:

Table 4: Percentage of Effort by Phase

<table>
<thead>
<tr>
<th>Waterfall Activity</th>
<th>% Effort</th>
<th>Spiral Activity</th>
<th>% Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan/Requirements</td>
<td>8%</td>
<td>Risk Assessment</td>
<td>5%</td>
</tr>
<tr>
<td>Product Design</td>
<td>18%</td>
<td>Requirements Analysis</td>
<td>5%</td>
</tr>
<tr>
<td>Detailed Design</td>
<td>25%</td>
<td>Product Design</td>
<td>20%</td>
</tr>
<tr>
<td>Coding/Unit Test</td>
<td>26%</td>
<td>Detailed Design</td>
<td>17%</td>
</tr>
<tr>
<td>Integration/Testing</td>
<td>31%</td>
<td>Coding/Unit Test</td>
<td>26%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Integration/Testing</td>
<td>22%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lessons Learned</td>
<td>5%</td>
</tr>
</tbody>
</table>
Verification and Validation of Model

Verification of the Incremental Spiral Model was straightforward since it was based on the IEEE 12207 model that was previously verified and validated. Changes to the original model were made incrementally and the model was verified to run without issues and to produce reasonable results. Final verification was accomplished by forcing the incremental spiral model to behave as a waterfall model and comparing results with that obtained from similar runs of the IEEE 12207 model. The output data from both models was similar when the spiral model was limited to one increment and one spiral, thereby reducing it to a similar process. Expected variances between the model outputs were explained due to the additional steps of risk assessment and lessons learned and % overall effort differences by lifecycle type.

The primary techniques for model validation are: develop a model with high face validity; test the assumptions of the model empirically; and test the model for reasonableness of output to actual project being modeled. The model was based on a spiral lifecycle approach that was successfully used at NASA’s Marshall Space Flight Center. This, along with the use of data from the respected Software Engineering Laboratory, helps to ensure face validity since the model is representative of past projects in the NASA environment. The assumptions of the model were tested using data from the literature and an actual NASA project. Finally, the model provided reasonable output when compared to actual data from the NASA project.
Cost and schedule estimation for large software development projects is historically inaccurate. Popular estimating models have been shown to be only within 25% of actual costs for 50% of the time (Ferens and Christensen 1998). Using a simulation tool, this paper presents an estimation approach that illustrates the effects of normal working dynamics on the cost of a large software development project throughout the project’s evolution. Simulation models can be used to communicate the uncertainty and complexity of the development process and can provide a check on other estimating methods that may be used.

The ability to obtain an accurate estimate of an entire project prior to its start is unfortunately unrealistic. And yet, cost and schedule estimates are necessary as management commits to funding such projects or bidding on a job. Simulation models are typically used to analyze the effects of process changes, and not for developing initial cost and schedule estimations. This paper will describe how simulation models can be used for this purpose and will show the benefits that can be obtained by using simulation as an estimation tool. The tool will illustrate the difficulties management faces in forecasting budgets at the beginning of a project and may encourage more realistic approaches to budgetary planning including phased funding. This simulation tool will also monitor changes in costs as the project evolves.

The task of cost estimation for project managers of software development projects becomes more and more daunting as the size and complexity of the project increases. Complex software development projects are likened to pure research and development projects, with all of
the inherent difficulties of managing and planning for work that is innovative and unique and that has uncertain requirements (Abdel-Hamid and Madnick 1991).

Planning purposes require that an estimate be developed at a time in the project when the values of key parameters such as product size and staff capabilities are unknown. This makes it unrealistic to provide an accurate estimate. Even if the values of key variables could be known with certainty at the beginning of a project, software activities are labor intensive and prone to all the complex and dynamic factors which affect human performance. Therefore, software development is not a deterministic activity and an estimate will require adjustments throughout the project until all the variables are known.

The initial estimate for a project is the most difficult and least accurate since there is less data available. Different tools and techniques for developing software project estimates exist, but none are guaranteed to give an accurate estimate. Often, though, the initial (and highly uncertain) estimate becomes the official estimate for the entire project and is used to judge whether or not the project is successful.

Trying to obtain a precise estimate at a very early stage in a project has lead to the use of techniques that do not depict uncertainty and complexity of the factors. Human nature prefers a single number for an estimate as opposed to a range of numbers, even though a range estimate will have a much higher chance of including an accurate value (Boehm and Fairly 2000). Much of the research work carried out in the software cost estimation field has been devoted to algorithmic models such as COCOMO and yet, methods that rely on expert judgment are still the most commonly used approaches (Agarwal and Kumar 2001). Research has shown that estimates based on expert judgment can be more accurate than those produced with analytical tools, especially if empirical data is used as a guide (Johnson, Moore et al. 2000).
Expert judgment approaches rely on experience on past projects and published industry averages. Average data does not tell the whole story and although past projects may seem similar, they will not have the same development costs, since estimates based on past experience do not account for changes in environments, politics, or organizations (Abdel-Hamid and Madnick 1991). In addition, historical data and experts’ memories of the past can be tainted. Even though expert judgment is the most often used technique, empirical software estimation models such as COCOMO are still widely used. These tools provide rigor to the estimating process, but the portability of these tools to different environments than the tool was developed for comes into question. The developer of COCOMO, Barry Boehm, admits that COCOMO is not right for every development environment (Boehm, Abts et al. 2000).

In essence, none of the approaches or tools available today can estimate the true cost of software with any high degree of accuracy early in a project. Managers should be presented with a technique that identifies risk and uncertainty based on the seemingly random nature of the variables and the complexity of the project system. Although managers must commit to a budget number and schedule, they should not be given a false sense of confidence in a point estimate. Adequate management reserves and phased funding should be considered to account for uncertainty, especially for larger and more complex software development projects.

The next section will describe the advantages and disadvantages of some commonly used expert judgment techniques and algorithmic models. This will be followed by a description of an estimation approach that uses a simulation process model in conjunction with commonly used techniques.
Commonly Used Estimating Techniques

Top-down and bottom-up are common approaches that are based on the judgment of experts. Often, more than one expert is used, as is done with the Delphi approach. In this case, individual estimates are developed by each expert. The individual estimates must fall within an acceptable range or the group meets until a consensus is reached. The top-down approach is often used for very early estimates since it uses global properties of the software product to develop a course estimate. The bottoms-up approach provides a more detailed estimate by dividing the project into modules, estimating for each module, and then rolling up module estimates to develop an overall project cost. Effects of environmental factors and process specifics are not accounted for in either approach. The uncertainty of the estimate is not quantified, but may be accounted for by requesting contingency amounts based on standard percentages. An advantage of these types of approaches is that experts can highlight unique strengths and weaknesses of local organizational characteristics (Agarwal and Kumar 2001).

COCOMO is a widely used algorithmic model. There are two versions of COCOMO, the original COCOMO 81 and the newer COCOMO II. The original COCOMO model was published in 1981 (Boehm 1981) and provided a straightforward model that predicts the effort and duration of a project based on inputs related to system size and 15 cost drivers that are believed to affect productivity. COCOMO 81 is still widely used, although it is only suited for a waterfall type process. The goal of COCOMO II is to provide accurate cost and schedule estimates for the newer lifecycle processes such as incremental and spiral (Boehm, Abts et al. 2000). COCOMO II defines three different models for cost estimation based on which phase of the life cycle the tool is being used. The models are intended to provide increasingly accurate
estimates, based on increasingly more detailed data that becomes available as progress is made through the software development cycle. For early prototyping and feasibility analyses stages, the Application Composition is the best model to use. The estimating equation is a simple linear relationship of object points and domain complexity. The Early Estimation Model is best for phases where architectures are explored or incremental strategies are developed. The Early estimation model uses lines of code for sizing and a coarse-grained set of seven cost drivers. Once the project is ready to develop and sustain a fielded system, the Post-Architecture stage is appropriate. The detail of this model is equivalent to COOMO 81 and uses seventeen multiplicative cost drivers and five exponential scale factors as a refinement of the COCOMO 81 development modes.

Even though COCOMO is formula based, a great deal of subjective input is involved in qualitatively rating attributes and selecting a suitable multiplying factor. There are many parameters that need to be uniquely rated from very low to very high and this is not easily done. For instance, how does a user consistently delineate the point between very low and low for a particular parameter? This type of model does not account for uncertainty that exists when selecting multiplying factors for the different parameters. The model does provide some insight into the complexity of the estimation problem due to the existence of so many parameters. Straightforward formulas, however, can give the wrong impression that an exact estimate can be developed just by “plugging” in data. Decision makers want to see an exact number, but need to understand that the complex dynamic interactions and process specifics that are not captured by this type of model but that have great impact, make it unlikely that a single number estimate will be exact.
The predictive accuracy of algorithmic models has been shown to be at most within 25 percent of actual cost and schedule one half of the time (Ferens and Christensen 1998). Some authors claim that a model’s predictive accuracy can be improved by calibrating the default parameters to a specific environment using historical data (Kemerer 1987; Boehm, Abts et al. 2000; Boehm and Fairly 2000) although this supposition has been debated (Ferens and Christensen 1998). The newer version of COCOMO, COCOMO II, is better suited to match modern day lifecycle practices, but its database is still comprised of diverse projects from numerous organizations and definitions and assumptions may not be the same for every development environment. It is believed that calibrating the multiplicative constant in the COCOMO effort equation using a linear regression approach with at least five data points is thought to account for differences in lifecycle activities and ratings of personnel factors. However, results of a Department of Defense Study showed that calibration does not always improve a model’s predictive accuracy and that reported accuracies can be overstated when studies fail to use a hold-out sample to validate the calibrated models (Ferens 1999). In addition, obtaining the proper amount of accurate historical data can be very difficult. The Software Engineering Laboratory was created to collect and analyze software development data from the NASA environment. In its 25 year existence, key factors that affect the complex process of collecting large amounts of useful project data have been identified: data collection is costly; it requires a rigorous process performed by a professional staff; developers may be reluctant to cooperate with data collection which reflects their performance (Basili, McGarry et al. 2002).
**Baseline Simulation Model**

Simulation models have been used to derive implications about the behavior of an organization through integration of the multiple functions of the software development process (Abdel-Hamid and Madnick 1991) and to quantitatively evaluate the performance of alternative software processes and process changes (Raffo 1996).

The Process Analysis Tradeoff Tool, PATT ©, is a discrete event process simulation model that was developed for NASA to assess the benefits of Independent Verification and Validation (IV&V) on the IEEE 12207 software development process (Raffo and Wakeland 2003). The tool is intended to enable adaptation to multiple projects and IV&V techniques. The model uses industry average data for input variables such as product size, productivity (LOC/Hr), and defects (per KSLOC). The user provides per cent of overall effort that should be allocated to each process step as well as the number of desired staff for each step. The model outputs the size, effort, rework effort, entire process duration, average duration, number of injected defects, detected defects, and corrected defects.

**Model Input Data**

The use of probability distributions for key variables such as size, productivity, and defects is a truer model of reality, especially in the early stages of a project. The model’s outcomes will be driven by random variables drawn from the probability distributions. Numerous runs of the process with different random numbers will provide the most meaningful information and will allow for the calculation of confidence intervals for each quantity of interest.
The Software Engineering Laboratory (SEL) began collecting data for NASA flight software development projects at Goddard Space Flight Center in 1976 and served as a major resource in software process improvement activities. (Basili, McGarry et al. 2002). Extensive project and product data was collected for over 200 projects and is available to the public (SEL 1997). It is recognized that the collection, analysis and retention of historical data for software development needs to be increased (Fairley 1992), but the process of doing so is complex, costly and time-consuming. Therefore, many organizations do not have adequate amounts of reliable data at their disposal. Data from the SEL has served as the basis of many software development “rules of thumb” concerning lifecycle activities and defect generation activities. The SEL data is especially appropriate for projects developed in the NASA environment with its stringent testing and reliability requirements.

As mentioned, the entire set of project data collected by the SEL is available. For the purposes of this research, subsets of the data that were collected and organized by the NSF Center for Empirically Based Software Engineering (CeBASE 2005) were used. Probability distributions were fitted for productivity data and defect injection data.

The uncertainty that exists in estimating the size of the final software product is also very important since size has been shown to be the key factor, followed by the effort adjustment multipliers, in models such as COCOMO (Musilek and Pedrycz 2002). It is important to also point out that the relationship between cost and system size is not linear and that cost actually increases exponentially as size increases (Abdel-Hamid and Madnick 1991). Experts claim that the problem of how to accurately size software is the greatest roadblock to improving estimation (Boehm 1981). There is a great deal of debate over how to best measure the size of a product. Lines of Code and Function Points are common methods for estimating the size and yet neither
accurately captures the quantity of knowledge that must be obtained and the difficulty of obtaining such knowledge (Armour 2002). Still, LOC continues to be the most widely used method of sizing a software project, especially for government and military projects and it is the variable that is used in the COCOMO equations. It is difficult to know with any certainty the size of a project early in the life cycle, even if similar projects are used as a basis for comparison. The problems of incomplete, vague requirements as well as requirements creep are almost always present for software development projects and will cause the size of the project to change over its development life cycle.

In order to capture the impact and uncertainty of sizing a software development project, a uniform distribution is used for the estimated size of the product. Figure 11 is adapted from a figure given in *Software Cost Estimation with Cocomo II* and portrays how much size can be underestimated at different points in the project lifecycle. For example, at the very early stage of a project where there is a concept of operations but no firm requirements, a size estimate can be off by a factor of 2. This means that an estimate at this point in the project of 1 Million LOC could actually end up being as high as 2 Million LOC by the end of the project. Therefore, using the figure in Exhibit 1 as a guide for size estimates at specific lifecycle phase such as concept of operations, a size estimate of 1 Million LOC would equate to the following distribution: Uniform (1 Million, 2 Million). The parameters of the distribution will change based on the estimate and the phase of the lifecycle for which the estimate is developed.
NASA Project

NASA’s software development projects are often large and complex due to the mission critical nature of their business. With human life and billion dollar payloads at risk, the NASA development environment is more stringent and complex than that of many other software development industries. For example, a bug in a windows product can cost money and customer dissatisfaction, but an error in launch processing software can lead to loss of life and vehicle.
NASA faces all of the same estimating difficulties discussed thus far. It is not uncommon to have managers who underestimate costs and then must continually request new funding at the risk of project cancellation. In fact, the U.S. General Accounting Office has criticized NASA managers for the past decade for failing to create realistic budgets for new projects. A recent government watchdog analysis showed that a majority of 27 recent projects was found to have costs that were very different from the initial estimates, some by as much as 94% (Asaravala 2004).

In order to explore the benefits of using simulation for cost estimation on large, complex software development projects, a real NASA project will be used as the subject of this study. The NASA project that will be evaluated with this approach involved the design and development of a new launch control system for the Space Shuttle. The project was cancelled after seven years and experienced cost overruns and schedule slippages throughout its history. The project faced many of the following common problems that make software development cost estimation especially difficult: costs and schedules are pre-determined by an outside source, the software development process is not fully understood or analyzed, requirements are not well-defined and prone to changes, new projects are almost always different from past ones, and software practitioners do not collect enough data from past projects (Agarwal and Kumar 2001).

Different estimating techniques were used to develop estimates for the project. This paper will focus on the project’s early estimates and will outline how the simulation tool could have been used during this timeframe as well as throughout the project.

The technique of estimation by analogy was first used to identify a rough cost number that would serve as a budgetary placeholder until a more detailed budget and schedule could be developed. Even at this very early point, there was an expected acceptable cost and schedule if
the project was to be approved. The estimate showed a project timeframe of five years and a total project cost that included approximately 1400 labor years of effort. Still, this very high-level cost and schedule budgetary placeholder would become the project’s official budget and schedule. The more detailed estimation techniques that were used after this initial estimate would all give the same results. History would show that the budget and schedule were not realistic and attainable.

Project personnel would become uneasy with the budget and schedule after a more detailed bottoms-up estimate was completed and yet there was perceived pressure to accept what was already in the budget. As mentioned, an initial estimate can often become the official budget even if that was not the intended purpose when it was developed.

**The Approach**

The information known at the time of the first early estimates will be used with the simulation approach to analyze the benefits of such an approach.

1. **Utilize Software Development Process Model**

   A graphical representation of a software development lifecycle process is useful when educating decision makers on the inherent complexity which makes large, complex software development projects so difficult. The model allows for the user to understand the flow of work that needs to be accomplished and to analyze the impact of things such as rework through graphic display of the software product moving through the lifecycle steps. Software
development process models demonstrate how the process works and can be used to highlight potential problem areas such as rework due to defects. The baseline software development process model used for this work was set up to use single values for key parameters such as size and productivity in order to analyze the effects of process changes.

2. Capture Uncertainty for Key Parameters by Using Probability Distributions

Three key parameters that greatly affect the cost and schedule of a software development project are size of the product, productivity of project personnel, and defect rates. It is possible to find average values for productivity and defect rates in the literature, but it is important to point out that these parameters are subject to many influences that can greatly vary the value of the parameters throughout the course of a project. A great deal of uncertainty exists when trying to estimate values for these very important parameters before a project begins and early in the project. Therefore, the use of probability distributions will allow for developing range estimates considering the uncertainty that exists before a project begins.

It is desirable to have data from similar projects and environments for developing data distributions in order to obtain credibility for using the data as inputs to the simulation model. If adequate data is not available, the parameters for reasonable distributions can be estimated.
3. Run Model and Obtain Confidence Intervals for Effort and Schedule

The following is sample output data that is displayed on the main screen of PATT for each run set:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Mean</th>
<th>Standard Dev</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>4812.8</td>
<td>55</td>
<td>KSLOC</td>
</tr>
<tr>
<td>Effort</td>
<td>21040.5</td>
<td>1048.2</td>
<td>Person Months</td>
</tr>
<tr>
<td>Rework Effort</td>
<td>1682.2</td>
<td>129</td>
<td>Person Months</td>
</tr>
<tr>
<td>Duration</td>
<td>91.7</td>
<td>11.2</td>
<td>Months</td>
</tr>
<tr>
<td>Avg. Duration</td>
<td>73.9</td>
<td>9.5</td>
<td>Months</td>
</tr>
<tr>
<td>Inj. Defects</td>
<td>103277.3</td>
<td>3582</td>
<td></td>
</tr>
<tr>
<td>Det. Defects</td>
<td>94093.6</td>
<td>3149</td>
<td></td>
</tr>
<tr>
<td>Cor. Defects</td>
<td>93866.7</td>
<td>3144</td>
<td></td>
</tr>
<tr>
<td>Latent Defects</td>
<td>9410.6</td>
<td>526</td>
<td></td>
</tr>
</tbody>
</table>

Figure 13: Sample PATT Output Data

More detailed data on parameter values for each replication can be found in the output database. This data can be used to calculate confidence intervals for the two primary estimation parameters of effort and duration.

For example, begin with five replications of the model and calculate 95% confidence intervals for effort in person-months and schedule in months.

Use the following equation for confidence intervals (Kelton and Sadowski 2004):

$$\bar{X} \pm t_{n-1,1-\alpha/2} \frac{s}{\sqrt{n}} \quad \text{Equation 4: Confidence Interval}$$
<table>
<thead>
<tr>
<th>Effort (person-months)</th>
<th>Duration (months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>22722.77</td>
</tr>
<tr>
<td>Standard Dev.</td>
<td>1056.2</td>
</tr>
<tr>
<td>Half-width, $\alpha=0.05$</td>
<td>1311.23</td>
</tr>
<tr>
<td>95% Confidence Interval</td>
<td>[21411.5, 24034]</td>
</tr>
</tbody>
</table>

Figure 14: Example Data from Five Replications

More replications will reduce the half-width of the confidence intervals because increasing the sample size will decrease the variability of the sample mean. This is desirable because the error for the point estimate for duration is greater than 15%. The following approximation can be used to decide how many replications are needed if it is assumed that there will be more than 30 replications (Kelton and Sadowski 2004):

$$n \approx n_0 \frac{h_0^2}{h^2}$$

Equation 5: Approximation for number of replications

$n_0$ = number of initial replications, $h_0$ = initial half-width, $h$ = desired half width

In order to reduce the half width so that the point estimate error is smaller and more acceptable (let’s say 5%), the equation is solved for $n$:

$$n \approx 5 \frac{13.9^2}{5^2} = 38.6$$

Therefore, the model should be run again with 40 replications. When presenting the data, the top half of the confidence interval should be focused on in order to deter the desire to accept the lower bound of the interval in order to meet the lowest possible cost and schedule. This will take into account other factors that may not be accounted for and will remind decision makers
that all will never go as planned and that Murphy’s Law is alive and well when it comes to large, software development projects.

4. Compare Model Results with Other Estimating Techniques

The model results can be used as a sanity check for estimates developed with other techniques and tools. If another technique gives an estimate that is not included in the confidence interval, then the simulation model should be used to further analyze and question an estimate that is low. Ideally, estimates should fall within the upper half-width of the confidence interval in order to allow for those unplanned and unfavorable events that will occur during the course of every big project.

5. Use Model Results to Debate Unrealistic Budgets

If a budget is set with values that do not fall within the confidence intervals produced by the simulation model, then the model should be used as a tool for debate. The key input parameters can be varied to show the impacts of such variation. The model should be run with animation so that decision makers can visualize the process and reasons for its complexity.

6. Update Model with Actual Project Data as Project Evolves

Actual project data can be used as inputs to the model as the project evolves. Size will be known with better certainty later in the project as will productivity and defect rates. It is still important to remember that many factors that affect these parameters can continue to change and
therefore to affect the parameter values. Still the model can be used to analyze effort and schedule to complete based on actual project values to date. The model can also be used to study problem areas and the effects of potential solutions.

**Approach Applied to Project**

The NASA project documentation shows that there were two official estimates before the project started. The first was an estimate by analogy and the second a bottoms-up estimate. Both estimates officially resulted in the same total cost and schedule for the project. A review of interim bottoms-up estimate documents and personal notes show that the total project cost values were lowered for the final and official estimate. Even at this very early point in the project, there was great pressure to develop estimates that matched an acceptable budget and schedule. The budget was set for a cost of approximately 1400 labor years of effort and five years for the schedule.

**First Estimate**

1. **Baseline Model**

   The earliest estimated size for the project was 1.4 million lines of code and this was based on a rough analogy with previous projects. At this point in time, no decisions had been made on an acceptable concept of operations, architecture, or lifecycle process for the project.
Previous similar projects had used a structured waterfall process so it is reasonable to utilize the baseline PATT discrete event process model.

2. Data Distributions

Since there is no actual project data at this point, data from the Software Engineering Laboratory will be used to develop appropriate distributions for use in the model.

Size: Referring to Figure 1, the size estimate can be underestimated by up to a factor of 4 at this very early point in the lifecycle. Therefore, a uniform distribution with parameters 1.4M LOC and 5.6M LOC will be used in the simulation. Note: An interesting fact to point out is that the estimated size of the project at the time it was cancelled was 5.8M LOC.

Productivity: Productivity data for over 140 projects from the Software Engineering Laboratory was analyzed and fitted with distributions using “best fit” software. The following distribution was selected for productivity: Erlang (1.36, 3). For different environments where there is not adequate historical data, a triangular distribution could be used and the parameters approximated by considering the minimum, maximum, and most likely values. Summary reports from the SEL state that many projects experienced productivities that ranged from 3 to 5 LOC/Hr with the average being closer to 3. Some very high productivities (8 LOC/Hr and higher) were experienced by a small class of ADA projects with high reuse (SEL 1993). Based on this information, an example triangular distribution with minimum 1 LOC/Hr, maximum 5 LOC/Hr and most likely 3 LOC/Hr could be used.
Defect Rates: Values for defect insertion rates that range between 10 and 60 defects per thousand lines can be found in the literature (CeBASE 2004) with the smaller rates observed for projects that utilize disciplined and structured software engineering practices. The SEL data was used to analyze the number of defects inserted per thousand lines of code for different phases in the lifecycle. The PATT model is set up to accept six different types of defects, three of which relate to the lifecycle phases of requirements, design, coding, and testing. The other two types of possible errors are from bad fixes and documentation errors. Four distributions were developed for the phases of requirements, design, coding and testing. These are as follows:

Table 5: Defect Injection Probability Distributions

| Requirements Defect Injection | Lognormal (2.62, 7.1) |
| Design Defect Injection       | Lognormal (17.13, 73.35) |
| Coding Defect Injection       | Weibull (28.39, 0.81)   |
| Testing Defect Injection      | Exponential (40.9)      |

A nominal value of 30 defects per thousand lines was used for bad fixes and documentation errors, since this type of defect data was not available for distribution fitting.

For development environments where there is not adequate data to fit distributions and for which the SEL data is not appropriate, lognormal distributions can be used with estimates on the mean and standard deviation for parameters. Previous work demonstrated that lognormal distributions are appropriate in software process simulation modeling (Raffo 1996). Industry averages can be used for selecting parameters of the distribution.
3. **Run Model.**

The model was run for five replications with the following results for size, productivity, and defect injection rates:

Based on the initial run set of five replications, the 95% confidence intervals for effort and schedule are:

**Effort:** 16,705 +/- 7536.2 Labor Months

**Duration:** 68.1 +/- 37.24 Months

These are not very useful confidence intervals since the half widths are too large.

In order to obtain a smaller half-width, the model needs to have a substantially higher number of replications.

With the desire of obtaining less than 10% error on the point estimates for both parameters, the number of replications is solved for by the following:

**Effort:**

\[ n \geq 5 \left[ \frac{7536^2}{1670^2} \right] = 101.8 \]

**Duration:**

\[ n \geq 5 \left[ \frac{37^2}{7.2^2} \right] = 140 \]

Therefore, run the model for 150 replications and calculate confidence intervals.
Here is the PATT main screen table of results:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Mean</th>
<th>S.D.</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>4812.8</td>
<td>55</td>
<td>KSLOC</td>
</tr>
<tr>
<td>Effort</td>
<td>21040.5</td>
<td>1048.2</td>
<td>Person Months</td>
</tr>
<tr>
<td>Rework Effort</td>
<td>1682.2</td>
<td>129</td>
<td>Person Months</td>
</tr>
<tr>
<td>Duration</td>
<td>91.7</td>
<td>11.2</td>
<td>Months</td>
</tr>
<tr>
<td>Avg. Duration</td>
<td>73.9</td>
<td>9.5</td>
<td>Months</td>
</tr>
<tr>
<td>Inj. Defects</td>
<td>103277.3</td>
<td>3582</td>
<td></td>
</tr>
<tr>
<td>Det. Defects</td>
<td>94093.6</td>
<td>3149</td>
<td></td>
</tr>
<tr>
<td>Cor. Defects</td>
<td>93866.7</td>
<td>3144</td>
<td></td>
</tr>
<tr>
<td>Latent Defects</td>
<td>9410.6</td>
<td>526</td>
<td></td>
</tr>
</tbody>
</table>

The following table summarizes the results of a run set with 150 replications:

Table 6: Results from 150 Replications

<table>
<thead>
<tr>
<th>Effort</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>1416 Labor Years</td>
</tr>
<tr>
<td>Half-Width</td>
<td>65 Labor Years</td>
</tr>
<tr>
<td>Confidence Interval</td>
<td>[1351, 1481] Labor Years</td>
</tr>
</tbody>
</table>
4. Compare Results with Other Estimating Techniques

The estimate by analogy led to a budget of 1400 labor years and 5 calendar years. The simulation model produces values that demonstrate that there is substantial risk in accepting this budget. The model can be used to show that many factors must be considered and that these add to the complexity, time and cost for the project. For instance, decision makers need to be made aware of the impacts of defects and rework on the project, which can be substantial. The rework effort calculated using the simulation model is 8% of the overall effort. Also, the size of the project and the productivity of workers that have not yet been assembled cannot be estimated with any degree of accuracy. The model tries to account for this complexity and uncertainty. The visual display that lays out the waterfall process also adds value in considering aspects that are important and that can be of great impact.

5. Debate Unrealistic Budgets

The first area of concern should be the schedule. Fred Brooks states in his famous book (Brooks 1978) that more software development projects fail due to a lack of calendar time than all other factors combined. Capers Jones states that, “Once a project blindly lurches toward an impossible delivery date, the rest of the disaster will occur almost inevitably” (Jones 1998).

One of the top ten software management tenets states that software development schedules should not be compressed by more than 25% of nominal (Royce 1998). The initial schedule of five years is dangerously close to a 25% compression of the time estimate from the
simulation model. Therefore, decision makers need to be made aware that a five year schedule is very risky and that a longer development schedule should be given serious consideration and its impacts assessed at this very early point since NASA is always concerned with launch manifests being affected by delivery dates.

The estimate of 1400 labor years falls a little short of the values in the confidence interval. Management should be made aware that factors such as productivity and size can greatly impact this estimate. At this point in time, the size of the product will most likely vary from the estimated 1.4 Million LOC. The productivity distribution is based on a NASA development environment of cohesive development teams with low personnel turnover and in some cases, high reuse and development language advantages. It is appropriate to accept these conditions for this project’s development environment at this early point, but the environment could easily be different and these types of differences could substantially affect the productivity numbers. The benefit to closely analyzing a very early budget and emphasizing obvious risks such as schedule in this case is that decision makers do not become too comfortable with unrealistic estimates. The longer unrealistic numbers are considered as acceptable, the harder it becomes to change those numbers. Also, changes to budgets later rather than earlier (preferably before a project starts) more negatively affect the team’s reputation and management’s confidence in the ability to successfully complete the project.
Second Estimate

The objective of the Bottoms-Up Estimate was to develop a rough order of magnitude cost and schedule for a five year project. As part of this estimate, the system architecture was selected and a concept of operations was developed.

1. Baseline Model

The size of the software was estimated to be 3.8 Million LOC. A productivity rate of 1 LOC/Hr was used for the estimate. The bottoms-up estimate adheres to the original schedule of five years. The estimate summary states that the schedule is aggressive and success-driven, but does not recommend a longer schedule.

2. Data Distributions

There will be changes in the size and productivity distributions for this estimate based on what was known at this point in the project. The defect distributions will be unchanged from the values used for the first estimate. A Uniform Distribution with parameters 3.8M and 7.6M will be used for size. A triangular distribution with parameters of 0.5 for minimum, 3 for maximum, and 1 for most likely will be utilized for one run set and then compared to a second run set with the original productivity distribution of Erlang (1.36, 3).
3. Run Model

The model was run two times with 150 replications for each. Table 6 summarizes data from the runs:

Table 7: Results Using Different Probability Distributions

<table>
<thead>
<tr>
<th>Input Size</th>
<th>Input Productivity</th>
<th>Effort Confidence Interval (Upper Half)</th>
<th>Duration Confidence Interval (Upper Half)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run Set #1</td>
<td>(3800,7600)</td>
<td>Erlang (1.36,3)</td>
<td>[2322, 2384]</td>
</tr>
<tr>
<td>Uniform</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run Set #2</td>
<td>(3800,7600)</td>
<td>Triangular (0.5,1,3)</td>
<td>[2028, 2080]</td>
</tr>
</tbody>
</table>

4. Compare with other Estimating Techniques

The data from these two run sets should clearly send up a warning signal about accepting the established budget of 1400 labor years of effort and five years of schedule. The simulation model demonstrates the impact of the 3.8 million LOC size estimate and productivity estimate of 1 LOC/Hr. The established budget represents an unrealistic goal that is setting the project up for failure.
5. Debate Unrealistic Budgets

This approach would have allowed project members to raise concern over the established and unrealistic budget from the initial time it was presented. This second run of the model in conjunction with the detailed bottoms-up should add validity to describing the predetermined budget as impractical. Even if decision makers agree to an aggressive schedule and budget, the simulation model can be used to bound the aggressiveness and to instruct on reasons for concern.

Summary and Conclusions

Simulation process models can provide benefit to the estimation process for software development projects, although they are not typically used for this purpose. A simulation tool can be used to graphically portray the complexity of the development process and can be used to explore the effects of uncertainty in the key parameters of size, productivity, and defects. Simulation process models can be used in conjunction with other popular estimating methods to serve as a check on the validity of the estimate developed with these other techniques. It is necessary to develop estimates before a project begins, but it is also necessary to understand that a point estimate developed with many unknowns and uncertainty is not going to be accurate. There is the danger that unrealistic cost and schedule estimates agreed to in order to get a project started can become the official budget and schedule with no easy way of revisiting and changing them. The goal of this work has been to demonstrate the benefits of using a simulation model when estimating to allow for more realistic budget and schedule determination including an interval estimate to help focus on the uncertainty of the estimates.
The development process for a large software development project is very complex and dependent on many variables that are dynamic and interrelated. Factors such as size, productivity and defect injection rates will have substantial impact on the project in terms of cost and schedule. These factors can be affected by the intricacies of the process itself as well as human behavior because the process is very labor intensive. The complex nature of the development process can be investigated with software development process models that utilize discrete event simulation to analyze the effects of process changes. The organizational environment and its effects on the workforce can be analyzed with system dynamics that utilizes continuous simulation. Each has unique strengths and the benefits of both types can be exploited by combining a system dynamics model and a discrete event process model. This paper will demonstrate how the two types of models can be combined to investigate the impacts of human resource interactions on productivity and ultimately on cost and schedule.

Process simulation models, system dynamics models, and static cost models already exist for software development projects. Each of these tools has advantages and disadvantages and the appropriateness of each depends on the application. System dynamics models are useful tools for demonstrating the dynamic behavior of a project and are based on project variables and tasks as a whole with no process details or intricacies being captured. Process models, on the other hand, do provide great detail on the process and can be used to provide guidance on the
sequence of process steps and information flows and can also be used to analyze proposed process changes. In addition, process models can support management planning and control activities. This type of modeling, however, does not capture the interactions and structural relationships as effectively as system dynamics modeling. Therefore, it is desirable to combine information from both types of models in order to more thoroughly analyze a project. The combination of continuous and discrete models is present in the literature for only a couple of cases. Two examples of combined models can be found in the work of Martin and Raffo (Martin and Raffo 2000; Martin and Raffo 2001; Martin 2002) and Donzelli and Iazeolla (Donzelli and Iazeolla 2001). Martin added continuously changing sections based on Abdel-Hamid’s model to a discrete event process model and affected the discrete event clock to run continuously. Donzelli and Iazeolla This work will consist of a separate system dynamics software and discrete event process model in order to maximize the benefits gained from each. Users will be able to understand and experiment with the system dynamics model separately and the data of interest will be sent to the discrete event process model to affect it.

For this work, system dynamics will be used to analyze human resource issues such as experience levels and turnover. This information will be combined with a discrete event model of a waterfall lifecycle process model. The human resource area was selected because of its potential impact on a project, especially early on when managers may perceive staffing issues too optimistically or not at all. A simplified system dynamics model is used to make it easy to understand and to capture the key variables of interest. Existing hybrid models such as Martin’s are very powerful, but also very complex. Previous work has shown that smaller and less complicated models are better for presentation to those that are not familiar with process models and simulation (Madachy and Tarbet 2000). The goal of this work is to communicate to decision
makers the potential impact of turnover and experience levels on the cost and schedule of a project.

**Discrete Event Process Simulation Model**

The Process Analysis Tradeoff Tool, PATT ©, is a discrete event process simulation model that was developed for NASA to assess the benefits of Independent Verification and Validation (IV&V) on the IEEE 12207 software development process (Raffo and Wakeland 2003). The tool is intended to enable adaptation to multiple projects and IV&V techniques. The model uses industry average data for input variables such as product size, productivity (LOC/Hr), and defects (per KSLOC). The user provides % of overall effort that should be allocated to each process step as well as the number of desired staff for each step. The model outputs the size, effort, rework effort, entire process duration, average duration, number of injected defects, detected defects, and corrected defects.

The use of probability distributions for key variables such as size, productivity, and defects is a truer model of reality, especially in the early stages of a project. The model’s outcomes will be driven by random variables drawn from probability distributions. Numerous runs of the process with different random numbers will provide more meaningful information.

Productivity rates are highly variable and algorithmic cost estimation models such as COCOMO do not model the factors affecting productivity very well (Kemerer 1987). Major variations from constant productivity can occur, especially with system programming products that use hundreds of thousands of lines of code, built by multiple teams and several layers of management (Putnam and Myers 1992).
It is important to communicate to decision makers how delays such as the time it takes for inexperienced staff to become as productive as experienced staff can affect a project. These types of delays can have major impact and yet are not always formally considered. Research has shown that schedule overrun problems can be attributed to the interaction of manpower-acquisition policy and turnover in addition to software estimation accuracy (Abdel-Hamid and Madnick 1991). Even when managers are aware of such delays, studies have shown that it is difficult to deal with the delays without tools that help to develop an adequate mental model of the dynamics of the system (Sengupta and Abdel-Hamid 1999).

Productivity is very dependent upon the skill and availability of the workforce for a project. Experienced staff will be more productive than inexperienced staff, and it is unrealistic to expect that all the staff on a particular project will be experienced on day one. Inexperienced staff must be trained and assimilated into the project environment and this takes time. Turnover of employees is an issue that will affect staffing and ultimately, productivity. Turnover rates as high as 34% are often seen on software development projects (Abdel-Hamid and Madnick 1991). As experienced employees leave a project, qualified new employees must be found and hired or transferred to the project. These inexperienced workers must now be trained and assimilated before their productivity levels can match those of experienced project personnel.

The interrelationships between such staff-related variables are best captured with system dynamics. The human resource sub-system of the Abdel-Hamid and Madnick system dynamics model of a software development project was used as a guide for adding continuously changing staff levels to this model (Abdel-Hamid and Madnick 1991) and so was the combined model developed by Martin (Martin 2002). Martin developed a combined model by integrating the entire Abdel Hamid and Madnick system dynamics model of the software development
environment with a discrete event model of the standard ISPW-6 software process. This work showed the benefits of combining continuous and discrete event models.

**Addition of Continuous Simulation to Model**

In the system dynamics model, resources are divided into experienced and inexperienced groups and the time-changing levels will be derived based on a turnover rate and assimilation rate. The ratio of experienced staff to total staff will be calculated and then sent to the discrete event model to affect productivity.

The System Dynamics model was created in Vensim software. Vensim models graphically display the connections and feedback loops of the system. It is possible to instantly see simulation results for all variables on the screen to view more detailed results of any selected variable of interest with different analysis tools. Figure 14 shows the Vensim display of a simplified human resource system dynamics model for software development.
Figure 15: Human Resource Model in Vensim

This simplified version of the Abdel-Hamid and Madnick Human Resource model (Abdel-Hamid and Madnick 1991) assumes that the organization is willing to hire and that there is no delay in hiring.

The literature suggests an average productivity of 3.5 LOC/Hr (SEL 1993), but as previously mentioned, there are many dynamic and interacting factors that can cause this value to vary. It is especially difficult to accurately estimate the productivity of a project’s staff before a project begins because the availability and skill of the workforce is not known. If a certain staff level and experience level is assumed, this can change throughout the course of a project due to turnover and hiring practices and it is crucial to consider the potential impacts of these on productivity and ultimately, cost and schedule.

The desired output from the system dynamics model is the number of experienced personnel available throughout the project. This number will be used to develop a ratio
multiplier to productivity and will be used in the discrete event software process model. It is
assumed that an inexperienced person is 50% as productive as an experienced person. Since a
ratio of 1 for \( \frac{\#\text{experienced staff}}{\#\text{total staff}} \) would equate to a productivity multiplier of 1 and
a ratio of 0 for this quantity would equate to a multiplier of 0.5, the following equation will be
used to calculate the productivity multiplier:

\[
productivity\text{multiplier} = 0.5 \left( \frac{\#\text{experienced staff}}{\#\text{total staff}} \right) + 0.5
\]

The system dynamics model will be used to calculate the productivity multiplier for each
day during the project and this data will be read into the discrete event model. Each time the
discrete event model attempts to draw a productivity number from the productivity distribution,
the time will be captured and the associated productivity multiplier will be used in the
calculation to affect the productivity draw.

The organization’s environment must be taken into consideration when selecting values
for the amount of time it takes for an inexperienced person to become experienced (assimilation
delay), the hiring delay, and the quit/transfer rate. A large NASA software development project
will be used as an example of how to combine data from the models. Considering the NASA
development environment, an assimilation rate of 6 months, a quit/transfer rate of 2 weeks, and a
zero hiring delay will be used in the model.

**Experimentation**

Data from a real NASA project will be used for this experimentation. The first cost and
schedule estimate was developed using previous projects for an estimate by analogy. No formal
requirements existed and the architecture had not been selected at this very early point. In order to capture and account for the large amount of uncertainty that existed at this point in time, probability distributions will be used for size, productivity, and defect insertion rates. The productivity and defect distributions will be based on data from the Software Engineering Laboratory (SEL). This organization collected software development data from the Goddard Space Flight Center Flight Dynamics organization for over 25 years (Basili, McGarry et al. 2002). The GSFC software development organization was responsible for the development of mission software and used NASA personnel as well as contractors. Probability distributions for defect injection rates are also based on SEL defect data. Since a large amount of the SEL’s data was readily available (CeBASE 2005) and since we are using a NASA project for analysis, the environments are similar and it is reasonable to use probability distributions based on this data.

The project’s estimate by analogy was based on a size of 1.4 Million LOC and the literature says that this size can be underestimated by a factor of 4 at this early point (Boehm, Abts et al. 2000). The following are the input parameters that will be used for the experiment:

- **Size**: Uniform (1400, 5600) KSLOC
- **Productivity**: Erlang (1.36, 3) LOC/Hr
- **Requirement Defect Injection**: Lognormal (2.6, 7.1) errors/KSLOC
- **Design Defect Injection**: Lognormal (17.1, 73.3) errors/KSLOC
- **Code Defect Injection**: Weibull (28.4, 0.8) errors/KSLOC
- **Test Defect Injection**: Exponential (40.9, 0) errors/KSLOC
- **Bad Fix Defect Injection**: 30 errors/KSLOC
- **Documentation Defect Injection**: 30 errors/KSLOC
The first run will use the discrete event process model only and will not provide any productivity adjustments based on staffing experience levels. Stated another way, the entire staff is experienced from day one and remains on the project for the entire time. The mean effort for this scenario is 16,705 person-months and the mean duration is 68 months.

In order to test the other extreme situation, it is assumed that the staff of 350 will always be inexperienced and therefore produce at a rate that is 50% of the probability distribution draws. The mean effort for this case is 34,002 person-months and the mean duration is 143.7 months. It is easily seen from this, that the mean effort and duration are more than doubled for this case.

Next, the system dynamics model is used to consider the effects of turnover. Turnover is set at 30% and the assimilation delay is set at 6 months. The entire staff of 350 is considered experienced on day one of the project. The following figures show the output for turnover, assimilation, and staffing levels.
Figure 16: Turnover vs. Time (Baserun)
Figure 17: Assimilation Delay vs. Time (Baserun)
Figure 18: Number of Inexperienced Personnel vs. Time (Baserun)
The productivity ratio is calculated and sent to the process model. The mean effort changes to 17,776 person-months and the mean duration equals 76 months.

A nice feature of Vensim is that the variables of interest can easily be changed and the model can be run with the effect of the changes displayed on top of the baseline run. If the turnover is lowered to 15%, the following results are obtained:
Figure 20: Turnover vs. Time (Experiment, Baserun)
Figure 21: Assimilation Delay vs. Time (Experiment, Baserun)
Figure 22: Number of Inexperienced Staff vs. Time (Experiment, Baserun)
Running the process model with this data leads to a **mean effort of 17,086 person-months** and a **mean duration of 70.8 months**.

The following table provides a summary of other scenarios and the effect on effort and duration:
Table 8: Summary of Effects of Turnover Rates

<table>
<thead>
<tr>
<th>Turnover</th>
<th>Starting # Experienced Staff</th>
<th>Starting # Inexperienced Staff</th>
<th>Effort (person-months)</th>
<th>Duration (months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>350</td>
<td>0</td>
<td>16705</td>
<td>68</td>
</tr>
<tr>
<td>15%</td>
<td>350</td>
<td>0</td>
<td>17086</td>
<td>70.8</td>
</tr>
<tr>
<td>30%</td>
<td>350</td>
<td>0</td>
<td>17776</td>
<td>76</td>
</tr>
<tr>
<td>15%</td>
<td>175</td>
<td>175</td>
<td>18098</td>
<td>89.2</td>
</tr>
<tr>
<td>30%</td>
<td>175</td>
<td>175</td>
<td>19839</td>
<td>89.5</td>
</tr>
<tr>
<td>15%</td>
<td>0</td>
<td>350</td>
<td>18497</td>
<td>83.8</td>
</tr>
<tr>
<td>30%</td>
<td>0</td>
<td>350</td>
<td>18949</td>
<td>86.9</td>
</tr>
</tbody>
</table>

Many different factors will affect the ultimate cost and schedule for a project. This work has shown the potential impact of the human resource issues of turnover and the experience level of the staff. The system dynamics model presents an easy to understand graphical representation of the interrelationships of key human resource factors that affect the experience level of staff. The discrete event process model utilizes this data to affect the productivity which in turn will affect the ultimate cost and schedule. A simplified system dynamics model is used for the purpose of demonstrating how the two types of simulation models can be used in conjunction to consider an important issue for large software development projects such as the impact of turnover on cost and schedule estimates..
CHAPTER SIX: A SOFTWARE DEVELOPMENT PROCESS MODEL OF A SPIRAL DEVELOPMENT PROCESS

ABSTRACT: There is a need for simulation models of software development processes other than the waterfall because processes such as spiral development are becoming more and more popular. The use of a spiral process can make the inherently difficult job of cost and schedule estimation even more challenging due to its evolutionary nature, but this allows for a more flexible process that can better meet customers' needs. This paper will present a discrete event simulation model of spiral development that can be used to analyze cost and schedule effects of using such a process in comparison to a waterfall process.

Software simulation models based on the traditional waterfall process exist as discrete event models and system dynamics models, but the literature points out the need to develop simulation models of other lifecycle processes. The traditional waterfall lifecycle development method has its drawbacks, so other lifecycle approaches are gaining popularity, especially for large and complex software development projects. The spiral development model was developed by Barry Boehm and is based on experience with large government software projects (Boehm 1988). The goal was to provide a model with greater flexibility that could better serve these types of projects. Boehm describes the spiral development model as a risk-driven process model generator that consists of a cyclic approach to incrementally implementing a system while decreasing the degree of risk (Boehm 2000).

The waterfall model is the traditional lifecycle development approach that was introduced by Winston Royce in 1970 (Royce 1970). This model consists of a sequential cycle of activities that include requirement analysis, design, coding, testing, and support. The spiral model can be thought of as a repeating waterfall model that emphasizes risk assessment and that is executed in
an incremental fashion. Each pass through the spiral model consists of risk assessment, requirements analysis, design, coding, testing, delivery, and evaluation. Figure 1 shows a graphical representation of a single increment of a spiral model.

Figure 24: Spiral Development Model

This process has been shown to be successful in a variety of environments, including NASA’s Marshall Space Flight Center (Hendrix and Schneider 2002). Figure 1 depicts the fact that the waterfall phases of requirements analysis, design, coding, testing and delivery are accomplished for each increment along with additional phases of risk assessment and lessons learned. Initial overall project planning and development of a concept of operations is accomplished prior to the first increment to establish high level requirements and the overall conceptual framework for the product. Detailed requirements evolve during the spiral portion of the lifecycle. The number of spiral passes that must occur for each increment depends on the areas of risk and the development state of the product. The goal of the increments is to provide the customer with limited, but useable operational capability. The customer will eventually get full operational capability after several increments.
A key difference between the waterfall and spiral models is that the waterfall model considers requirements to be fixed from the beginning of the project with fixed documents being produced as a result of each phase of the lifecycle. Therefore, this process is not flexible enough for some projects, especially when requirements are not known at the beginning of a project. Changes in requirements later in the process lead to major cost and schedule overruns, especially for very large projects. This approach can lead to other problems such as: delayed integration, late risk resolution, and focus on documents and review meetings as opposed to tangible increments of the product (Royce 1998). Therefore, more and more software development projects are following lifecycle models other than the waterfall model.

The spiral model is designed to be flexible and to evolve into other types of models such as evolutionary or even waterfall based on the results of risk assessments and where key risks exist (Boehm and Belz 1990). It is also considered product driven rather than document driven like the waterfall process (Royce 1998). The government and military are using this process more often in order to overcome the limitations of the more traditional processes. The US Dept. of Defense has determined that the spiral development model is the preferred method/process for software-intensive development lifecycles (Surber 2004).

Disadvantages of the spiral model include continual deferral of planned functionality in order to stay on schedule and within budget. The deferral of work can accumulate until an insurmountable amount of work is left for the end of project. This is known as the “Death Spiral” (Brown 2004). Projects that follow this type of development process will most likely cost more and take longer but should better meet customers’ needs and expectations.

A discrete event process simulation model of the spiral development lifecycle process will allow for evaluating different scenarios in projects using this type of approach. A process
model for spiral development can also enable analysis of the effects of using this type of approach versus the traditional approach in terms of effort and schedule. The waterfall approach assumes all requirements are known up front and yet this is often unrealistic. Changes to requirements will affect size and therefore, the cost and schedule of the project. In order to handle changes that occur as a project evolves, NASA’s Manager’s Handbook for Software Development recommends that a minimum of five re-estimates should be made after the initial estimate at key life cycle phase points (SEL 1990). This will allow for more accurate estimates to be developed as the project progresses and as more information becomes available. The handbook states that the uncertainty of an estimate will decrease from completely uncertain at the initial estimate to almost certain with the sixth estimate after system test.

Cost estimation is an especially difficult area of software development project management. The impacts of uncertainty in key areas such as product size, productivity, and defect injection rates can dramatically affect a project’s cost and schedule. The job of estimating becomes even more difficult when requirements are allowed to evolve throughout a project as is the case for a spiral lifecycle process. The same unknowns of size, productivity, and defect injection rates exist, but there is also the additional unknown of the number of spirals that will need to be completed before an incremental product is delivered. The evolutionary nature of the process allows requirements to change and this makes the job of estimating size even more difficult and uncertain.

Figure 25 shows the layout for a typical software development process simulation model of a waterfall type project.
An existing software development process simulation model will serve as the core for the spiral process simulation model. The Process Analysis Tradeoff Tool, PATT ©, is a discrete event process simulation model that was developed for NASA to assess the benefits of Independent Verification and Validation (IV&V) on the IEEE 12207 software development process which is a waterfall type process and which is represented by Figure 2 (Raffo and Wakeland 2003). The model typically uses industry average data for input variables such as product size, productivity (LOC/Hr), and defects (per KSLOC). The user provides number of resources, % of overall effort that should be allocated to each process step, and the number of desired staff for each step. The model outputs the size, effort, rework effort, entire process duration, average duration, number of injected defects, detected defects, and corrected defects.
Model Development

The approach to developing the spiral model was to add steps to the existing PATT waterfall model and to repeatedly run through the steps to represent increments for the entire project.

Figure 26 shows how PATT serves as the core of the spiral model.

![Figure 26: Use of PATT for Spiral Model](image)

Model Inputs

The following information is input to either the IEEE 12207 PATT model or the spiral model: size of product (lines of code), number of resources, productivity (lines of code/hour), defect injection rate (defects/ksloc), % of overall effort for each process step, and desired staff for each process step. Industry averages or theoretical probability distributions can be used for productivity and defect injection rate.
The following table provides guidelines for percentage of effort by phase that is based on several literature sources (Boehm 1981; SEL 1993; Boehm, Abts et al. 2000; Hendrix and Schneider 2002).

Table 9: Percentage of Effort by Phase for Waterfall and Spiral Processes

<table>
<thead>
<tr>
<th>Waterfall Activity</th>
<th>% Effort (Boehm 2000)</th>
<th>% Effort (SEL)</th>
<th>Spiral Activity</th>
<th>% Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan/Requirements</td>
<td>8%</td>
<td>12%</td>
<td>Risk Assessment</td>
<td>5%</td>
</tr>
<tr>
<td>Product Design</td>
<td>18%</td>
<td>8%</td>
<td>Planning/Requirements</td>
<td>5%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Analysis</td>
<td></td>
</tr>
<tr>
<td>Detailed Design</td>
<td>25%</td>
<td>15%</td>
<td>Product Design</td>
<td>20%</td>
</tr>
<tr>
<td>Coding/Unit Test</td>
<td>26%</td>
<td>40%</td>
<td>Detailed Design</td>
<td>17%</td>
</tr>
<tr>
<td>Integration/Testing</td>
<td>31%</td>
<td>25%</td>
<td>Coding/Unit Test</td>
<td>26%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Integration/Testing</td>
<td>22%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Lessons Learned</td>
<td>5%</td>
</tr>
</tbody>
</table>

Note that the % effort for a waterfall process based on data found in *Estimating with COCOMO II* totals to 108%. This is due to the assumption that all plans and requirements for a waterfall
approach will be completed prior to the beginning of a project. Data in the second % Effort column for waterfall is based on Software Engineering Laboratory data and totals to 100%. Data from Table 1 will serve as the % of overall effort input for the models.

Industry averages from the literature suggest an average productivity of 3.6 LOC/Hr (Jones 2000) and defect injection rates of 60 defects per thousand lines of code (CeBASE 2004). For this work, data from the NASA environment was used to populate the model for analysis. Productivity and defect injection rates tend to be lower for the NASA environment with an average productivity of about 3.2 LOC/Hr (SEL 1993) and defect injection rates of approximately 30 errors per KSLOC (CeBASE 2004). Since productivity and defect injection rates are affected by project factors that are very dynamic, the use of average values does not adequately account for the range of values of these key parameters. In order to better represent the range of values that can occur for productivity and defect injection rates in a similar development environment, probability distributions were developed and used in the models. These distributions are based on data collected during the 25 year history of the Software Engineering Laboratory (SEL 1993; CeBASE 2005). The following provides the distribution and parameter values used as inputs to the models for defect injection rates:

Table 10: Defect Injection Rates by Phase

<table>
<thead>
<tr>
<th>Phase</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Lognormal (2.62, 7.1)</td>
</tr>
<tr>
<td>Design</td>
<td>Lognormal (17.13, 73.35)</td>
</tr>
<tr>
<td>Coding</td>
<td>Weibull (28.39, 0.81)</td>
</tr>
<tr>
<td>Testing</td>
<td>Exponential (40.9)</td>
</tr>
</tbody>
</table>
A distribution of **Erlang (1.36, 3)** was used for productivity.

**Analysis**

In general, large NASA software development projects begin with a bottoms-up estimate prior to the start of a project. For this analysis, assume a staff of 350 was available. The bottoms-up estimate included size estimates for each increment that added up to 3.425 Million LOC. Funding was provided based on a total size of 3.8 Million LOC, 1400 labor years, and a schedule of five years. At this point, a Concept of Operations with high level requirements was completed. The goal was to follow an incremental development process that consisted of 10 increments and allowed requirements to evolve during the project.

Of interest is the question of how does a waterfall approach compare with a spiral approach, especially when developing early estimates. First, the IEEE 12207 process model and the spiral model will be run with the same input data so that the output effort and duration data from each can be compared. For this first analysis, the assumption will be that the size estimate is accurate (which would be highly unlikely at the beginning of a project and thus favors the waterfall model), so a total size of 3425 will be input to the IEEE 12207 model. Table 11 provides size for each increment of the spiral model:
Table 11: Size Estimate Per Increment

<table>
<thead>
<tr>
<th>Increment Number</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>675</td>
</tr>
<tr>
<td>4</td>
<td>700</td>
</tr>
<tr>
<td>5</td>
<td>650</td>
</tr>
<tr>
<td>6</td>
<td>200</td>
</tr>
<tr>
<td>7</td>
<td>325</td>
</tr>
<tr>
<td>8</td>
<td>300</td>
</tr>
<tr>
<td>9</td>
<td>175</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
</tr>
</tbody>
</table>

The spiral model will include an additional probability distribution for the number of spirals that must be completed within each increment. This will be set to a Uniform [1, 3] distribution.

Table 12 provides the output data from each type of model:
Table 12: Comparison of Outputs for Spiral and Waterfall Models

<table>
<thead>
<tr>
<th>Output</th>
<th>Waterfall</th>
<th>Spiral</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Duration</td>
<td>5.9 years</td>
<td>8.27 years</td>
</tr>
<tr>
<td>Mean Effort</td>
<td>1391 labor years</td>
<td>1520 labor years</td>
</tr>
</tbody>
</table>

This data shows that a budget of 1400 labor years and a schedule of five years are very risky for this project, even if a waterfall process is followed. This data also shows that a spiral process will take longer than a theoretical waterfall process and this should be expected, although it may not be considered when preparing a budget. The spiral process should take longer and cost more because the process is repeated and has additional process steps. This makes a one time estimate done at the beginning of a project very impractical. A rough estimate can be developed, but detailed phased funding should be considered so that more accurate estimates can be developed based on an incremental basis rather than for the entire project. The intended benefit of spending more time and money on such a process is that the user will get a better product and will get incremental functionality with each delivery.

Size growth due to requirements changes and unknowns can be extensive. The literature points out that very early size estimates are likely to be much lower than the actual final size of the project due to requirements changes and unknowns(SEL 1994; Boehm, Abts et al. 2000). This is often the case with either the waterfall or spiral process, even though requirements should not drastically change in a true theoretical waterfall lifecycle. Requirements evolution is a key part of a spiral process and therefore, plans for accommodating size changes should be considered when estimating a project that will follow such a process. This is a more realistic view of the situation for large, complex projects than a theoretical waterfall process.
Since the project was set up for ten increments, each increment will have a uniform size distribution with parameters: \( (\text{Increment Size Estimate, } 2 \times \text{Increment Size Estimate}) \text{ LOC.} \)

The uncertainty in the size at this point in the project when only high level requirements are understood is based on data from the literature (Boehm, Abts et al. 2000). Table 13 provides the size inputs to the model for each increment:

Table 13: Incremental Size Distributions

<table>
<thead>
<tr>
<th>Increment</th>
<th>Size (KSLOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Uniform [100, 200]</td>
</tr>
<tr>
<td>2</td>
<td>Uniform [250,500]</td>
</tr>
<tr>
<td>3</td>
<td>Uniform [675,1350]</td>
</tr>
<tr>
<td>4</td>
<td>Uniform [700,1400]</td>
</tr>
<tr>
<td>5</td>
<td>Uniform [650,1300]</td>
</tr>
<tr>
<td>6</td>
<td>Uniform [200,400]</td>
</tr>
<tr>
<td>7</td>
<td>Uniform [325,650]</td>
</tr>
<tr>
<td>8</td>
<td>Uniform [300,600]</td>
</tr>
<tr>
<td>9</td>
<td>Uniform [175, 350]</td>
</tr>
<tr>
<td>10</td>
<td>Uniform [50,100]</td>
</tr>
</tbody>
</table>

At the beginning of each increment, a draw will be taken from a probability distribution for the number of spirals that are to be completed and the total increment size will be divided by the number of spirals to provide the size for each spiral. This will be compared to the waterfall model. For this analysis, the waterfall model will be run with a size distribution of Uniform
and the % of effort in Table 9 that totals to 100% to represent a more realistic waterfall where requirements analysis is done as part of the project.

<table>
<thead>
<tr>
<th></th>
<th>Spiral</th>
<th>Waterfall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Duration</td>
<td>14.67 years</td>
<td>9.89 years</td>
</tr>
<tr>
<td>Mean Effort</td>
<td>2414.6 labor years</td>
<td>2356 labor years</td>
</tr>
</tbody>
</table>

This data also shows that the spiral process should cost more and take longer than a waterfall approach. The analysis also shows the significant impact size uncertainty has on a project’s cost and schedule for either lifecycle approach. A benefit of following a spiral process that consists of multiple spirals per increment is that particular risk areas can be resolved by expending only a portion of the increment’s budget. The spirals serve as a risk resolution plan that should enable deliverable functionality for each increment. The fact that this type of process will provide benefits but may take longer and cost more must be considered when developing estimates for a project that will follow this approach.

**Summary**

This work has developed a software development process model that enables assessment of an incremental or spiral lifecycle approach. The waterfall process provides for a structured sequential process, but this is often not realistic since requirements tend to evolve and phases of the lifecycle may need to be repeated. Because software development can be very complex and uncertain, iterative lifecycles are gaining popularity. Therefore, the effects of such a process need to be analyzed and understood. Using estimation data from the NASA environment, this work
has analyzed a project’s early cost and schedule estimate using both a waterfall and spiral approach. The results show that the cost and schedule estimates for a spiral process may be higher than for a waterfall process and this is in agreement with military experiences (Brown 2004). This type of process should provide a quality product that better meets users’ needs by allowing evolution of requirements and by providing functionality with each increment. This process emphasizes risk management and is designed to be flexible. This should lead to more realistic budgetary planning since it is obvious that requirements will change cost and schedule will be affected throughout the project. This work also shows that uncertainty in areas such as size, productivity, and defects should be accounted for when developing an estimate, no matter which lifecycle is selected. The spiral process model can be used throughout a project to analyze the project as more information becomes available. For instance, data from early increments can serve as inputs to the model and an estimate to complete based on this data can be assessed. More work can be done to refine the spiral model based on other projects’ experiences.
CHAPTER SEVEN: CASE STUDY ANALYSIS OF A LARGE, COMPLEX SOFTWARE DEVELOPMENT PROJECT USING SIMULATION

Large operational software development projects, i.e. those that are performed by NASA, are very difficult to manage. In general, software development projects are often over budget and behind schedule and the larger and more complex a project is, the greater the risk of cancellation. NASA’s software development projects are often large and complex due to the mission critical nature of their business. With human life and billion dollar payloads at risk, the NASA development environment is more stringent and complex than that of many other software development industries. This case study will investigate specific project management problems that were encountered by a NASA project that was cancelled prior to completion. The project will be evaluated using a simulation tool so that it is possible to identify the potential benefits the tool could have provided. Emphasis will be placed on the problem areas of cost and schedule estimates and the effects of different lifecycle approaches.

NASA has had success in the realm of large operational software projects such as the Space Shuttle Onboard Software project at the Johnson Space Center. In fact, this project is cited by the Software Engineering Institute as being level 5 on the Capability Maturity Model (CMM), the highest possible on the widely-used process maturity scale (Carnegie Mellon University/Software Engineering Institute 1994). However, numerous large development projects have been cancelled due to large overruns and schedule delays.

The following characteristics of software development give some insight into why it is so difficult to manage: the development of software is more labor intensive than the development of any other business product (Jones 2000) and therefore highly susceptible to factors that affect personnel and human performance; the primary deliverable of the software itself is an intangible
item that does not lend itself to easy determination of product status; the requirements and design for software projects tend to drastically change throughout the development cycle.

The Checkout and Launch Control System (CLCS) Project was a modernization effort to replace the aging Launch Processing System (LPS) at the Kennedy Space Center. This project followed two previous failed projects with similar goals. Project data in the form of reports, presentations, and personal notes were all made available for this study. Discussions with key personnel involved in planning and managing the project were also used to obtain data and background information.

The Checkout and Launch Control System (CLCS) project suffered from cost overruns and schedule slippages. Many large software development projects fail because of under-funding and lack of calendar time (Boehm 1981). Projects that face resource shortages whether in the form of funding, schedule, or personnel that are 50% lower than what is needed can be considered “death march” projects (Yourdon 1997). Based on this criteria and project data, CLCS can be considered a death march project. The project began with an initial baseline estimate of $206 million that grew to $400 million over the seven year time span prior to cancellation. Yourdon points out that death march projects are the norm and not the exception, so NASA is not alone in dealing with this type of project.

The project underwent major re-planning before it was eventually cancelled. Those involved with the project will tell you that there were many reasons the project was cancelled. The fact that this highly visible project had to repeatedly request extra funding and schedule certainly did not help the project succeed and probably made it more vulnerable to organizational politics. Therefore, a key area of investigation for this case study will be the estimation process used for the project and the effects it had on the project.
The initial budget and schedule for a project is tenuous, but often becomes the official mark against which the project is judged. Many software estimation tools and techniques do not provide an adequate means of capturing and portraying the level of uncertainty and complexity that exists for estimating for software development projects. Improvements are being made to automated cost estimation tools, but these tools still cannot provide an accurate estimate, especially when used early in a project (Ferens 1999).

Discussions with project managers on the project show that all felt that it would be reasonable to ramp up to 340 persons after one year and that the project could be completed in seven years. At the time, KSC was downsizing and it was estimated that 300 persons would be available to the project. There was pressure to show that the project could meet a pre-allocated amount of $225M over a five year period, with only 3.5 years of real development and the last 1.5 years of primarily sustaining and launch readiness activities. No one can ever know exactly how this predetermined amount came to be, but it was extremely optimistic and from the beginning put the project in a position of being a “death march” project with no real chance of successfully completing within budget and on schedule.

The following quote by Capers Jones captures the crucial nature of initial project planning and estimating:

The seeds of major software disasters are usually sown in the first three months of commencing the software project. Hasty scheduling, irrational commitments, unprofessional estimating techniques, and carelessness of the project management function are the factors that tend to introduce the terminal problems. Once
a project blindly lurches toward an impossible delivery date, the rest of the disaster will occur almost inevitably (Jones 1998).

The project faced many of the following common problems that make software development cost estimation especially difficult: costs and schedules were pre-determined by an outside source, the software development process was not fully understood or analyzed, requirements were not well-defined and prone to changes, the new project was different from past ones, and software practitioners did not collect enough data from past projects (Agarwal and Kumar 2001). Often, there is a predetermined budget that makes the estimation process an exercise in “gaming” so that the estimation techniques used provide the desired budget and schedule. No one can say for sure how the initial numbers for the CLCS budget and schedule came about, but there was great pressure to accept a budget and schedule that many felt were unrealistic and yet they felt powerless to reject.

**Project Background**

The Checkout and Launch Control System (CLCS) Project was a modernization effort to replace the aging Shuttle Launch Processing System (LPS) at the Kennedy Space Center. The project was considered a classic integration effort with a high degree of software development (4M lines of code). The CLCS project was officially initiated by a 1996 study that attempted to utilize the lessons learned from two previous cancelled LPS upgrade projects.

The project experienced some early successes, but the very tight cost and schedule limitations caught up with the project as the complexity of the deliverables increased and a series of delays and cost overruns occurred. Independent Assessments of the project and its processes
found that problems existed with schedule pressure, staffing, training, communications, and requirements uncertainty. The rapid start of the project, with the first delivery scheduled for just three months after the project start date, led to a lack of detail in the project planning. Unplanned events and lack of detailed requirements made the problem of an unrealistic schedule even worse. As a result, team morale suffered due to schedule pressures and heavy overtime and turnover became an issue. In addition, that actual staff levels and experience levels did not match what was planned. Staff training was found to be inadequate due to schedule pressures and a high attrition rate. The highly partitioned nature of the CLCS system architecture resulted in two separate development efforts, system software and application software, emerging from the project. The interaction between the groups was inadequate for insuring smooth final integration. The decision to keep requirements at a high level and to negotiate detailed requirements during each build was adequate for the first incremental deliveries, but became a problem as the project matured and inadequately defined requirements began to require rework. The situation was so bad that requirements were changing even through final testing of an increment. Cost overruns and schedule slippages made it necessary to completely re-plan the project. A new funding limit of $400 million was approved as was a final Operational Readiness Date of July 2006. In October 2001, project management decided that another project re-plan was necessary. Assessments for the estimate to complete at this time were significantly higher than previous estimates and it was highly probable that there would need to be a several month slip to the final launch-capable date. The decision to cancel the project was made on September 16, 2002.
Analysis

Simulation process models will be used to analyze the CLCS project from a cost and schedule perspective. Each run of the model will reflect a different level of information that was available at varying points in the project’s history. The goal of this analysis is to demonstrate how the use of simulation could have provided benefit to the project managers when they were faced with estimation and lifecycle decisions.

Early Project Analysis

A 60 day study was conducted in the summer of 1996 to develop an initial project estimate. An estimate by analogy was performed prior to this to serve as a budgetary placeholder and was developed based on experiences with other large software development projects. The estimate assumed a product size of 1.4 million lines of code, budget of $225 million, and a five year development schedule. Even though the more detailed bottoms up budget would estimate a much larger product size of 3.425 million lines of code, the budget of $225 million and the five year schedule would remain until the project was re-planned in 2000.

Everyone associated with the project recognized that this schedule and budget were very tight. The official rationale for accepting this risk was that the project would be a modernization effort of an existing system as opposed to creating a new system from scratch. The unofficial rationale for accepting this risk was that the cost estimates were based on available funding rather than engineering estimates of the actual projected costs and this was agreed to prior to the 60-day study. There was great pressure to accept this pre-established budget and therefore the official estimates were “tweaked” to match the available funding and schedule.
During the 60 day study, a suitable architecture was selected and it was decided that the project would be delivered in increments every six months. Initial requirements were defined at a high level in order to allow flexibility when refining them at a later time. This decision was made in response to lessons learned from previously failed projects. The goal of each incremental delivery was to provide additional system capability that builds on top of previously delivered capabilities. The budget and schedule assumed that it would be possible to quickly ramp up to an experienced workforce of over 340 persons, that COTS software development tools would be used extensively to reduce the amount of new LOC that needed to be written, and that there would be minimal growth in system requirements.

The software architecture consisted of system software, user application software, and gateways. The cost was based on size estimates of 1.4 million LOC for system software, 1.825 million LOC for application software, and 200 KSLOC for gateways for a total size of 3.425 million lines of code. Table 14 provides the size estimates by increment:
Table 14: Size Estimates for Each Increment

<table>
<thead>
<tr>
<th>Increment</th>
<th>Size (KSLOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>675</td>
</tr>
<tr>
<td>4</td>
<td>700</td>
</tr>
<tr>
<td>5</td>
<td>650</td>
</tr>
<tr>
<td>6</td>
<td>200</td>
</tr>
<tr>
<td>7</td>
<td>325</td>
</tr>
<tr>
<td>8</td>
<td>300</td>
</tr>
<tr>
<td>9</td>
<td>175</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
</tr>
</tbody>
</table>

Funding was provided based on a total size of 3.8 million LOC which accounted for 15% contingency on size growth. This is an unrealistically low amount of contingency, especially for size estimates developed this early in a project that will follow a lifecycle approach that allows requirements to evolve for each increment. This analysis will show that even if requirements and size did not change, the funding and schedule were very risky.

The first analysis will use a simulation model of the waterfall process. The assumption that the estimated size of 3.8 million LOC is not going to change will be reflected in the model with a deterministic value for size of 3.8 million LOC as an input. It will also be assumed that the staffing will be available immediately and that the experience level will be high.
Productivity and defect injection values can greatly vary throughout a project and these will impact the cost and schedule of the project. In order to properly capture these values, probability distributions were developed for each based on historical data from a similar environment. More information on this can be found in the article titled, “A Project Management Approach to Using Simulation for Cost Estimation on Large, Complex Software Development Projects” submitted to the Engineering Management Journal.

The following table lists the values for productivity and defect injection rates used for this analysis:

Table 15: Input Values for Productivity and Defects

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Productivity</td>
<td>Erlang (1.36,3) KSLOC</td>
</tr>
<tr>
<td>Defect Injection – Requirements Phase</td>
<td>Lognormal (2.62, 7.1) defects/KSLOC</td>
</tr>
<tr>
<td>Defect Injection - Design</td>
<td>Lognormal (17.13, 73.35) defects/KSLOC</td>
</tr>
<tr>
<td>Defect Injection - Coding</td>
<td>Weibull (28.39,0.81) defects/KSLOC</td>
</tr>
<tr>
<td>Defect Injection - Testing</td>
<td>Exponential (40.9) defects/KSLOC</td>
</tr>
<tr>
<td>Bad Fixes, Documentation Errors</td>
<td>30 defects/KSLOC</td>
</tr>
</tbody>
</table>

Table 16 provides the model results for this scenario:
Table 16: Results for Waterfall Model With No Size Uncertainty

<table>
<thead>
<tr>
<th>Metric</th>
<th>Mean Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>8.74 years</td>
</tr>
<tr>
<td>Effort</td>
<td>1608.5 labor years</td>
</tr>
</tbody>
</table>

The model shows that funding for 1400 labor years of effort and 5 years of calendar time is not going to be enough, even if the size estimate does not change. The literature suggests that a project that has its schedule compressed by more than 25% is doomed for failure. This project had too optimistic of a schedule even if requirements were known with absolute certainty and this was certainly not the case because of the lifecycle approach that was chosen.

Next, the model will be run with reasonable size variation. Based on the literature, a size estimate at this point can be low by as much as 50% (Boehm, Abts et al. 2000) (SEL 1990). Therefore, the model will be run again with a size distribution of Uniform [3.8 million, 7.6 million] to account for changes and unknowns.

The model results are as follows:

Table 17: Results for Waterfall Model with Size Uncertainty

<table>
<thead>
<tr>
<th>Metric</th>
<th>Mean Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>13.5 years</td>
</tr>
<tr>
<td>Effort</td>
<td>2408 labor years</td>
</tr>
</tbody>
</table>

130
In order to assess the affects of an incremental lifecycle that allows requirements to evolve, another process simulation model will be run. It will first be run with the assumption that the size values in Table 14 are accurate. The model results are given in the table below:

Table 18: Model Results of Incremental Model With No Size Uncertainty

<table>
<thead>
<tr>
<th>Metric</th>
<th>Mean Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>9.2 years</td>
</tr>
<tr>
<td>Effort</td>
<td>1549 labor years</td>
</tr>
</tbody>
</table>

Next, the same process model will be run, but each increment size will be input to the model as a uniform distribution with parameters [increment size, 2 X increment size]. This will more effectively portray that requirements, and therefore size, will change. The results of this scenario are below:

Table 19: Model Results of Incremental Model with Size Uncertainty

<table>
<thead>
<tr>
<th>Metric</th>
<th>Mean Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>14.5 years</td>
</tr>
<tr>
<td>Effort</td>
<td>2379 labor years</td>
</tr>
</tbody>
</table>

The model results show that the CLCS project was facing an impossible situation with a budget that supported 1400 labor years of effort and five calendar years of schedule. Also, the goal of delivering each increment every six months was not realistic when the sizes greatly varied for each deliverable. The table provides the model results for average duration times for each increment:
Table 20: Durations for Each Increment with Size Uncertainty

<table>
<thead>
<tr>
<th>Increment</th>
<th>Mean Duration (months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.8</td>
</tr>
<tr>
<td>2</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>37</td>
</tr>
<tr>
<td>6</td>
<td>8.25</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>14.3</td>
</tr>
<tr>
<td>9</td>
<td>5.2</td>
</tr>
<tr>
<td>10</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Also, there is a tendency to develop less challenging functionality up front and therefore, later increments would probably be more challenging in terms of coding, testing, and integration. Calendar pressure would lead project managers to delay functionality to later increments and this is a common issue with an evolutionary type process.
Analysis Based on Interim Deliverable Actual Data

Schedule slippages and cost overruns caused the project to be re-planned. An assessment team was brought together to review the project and to make recommendations to help the project get on the right track.

The first increment was considered a success because it was delivered on schedule, but it served as a test and did not produce any deliverable code. Pieces of the 3rd and 4th increments needed to be delayed, so the project was re-planned based on actual data to date. This next analysis will be based on data that was available in February of 1999 and will focus on the system software portion. This area was the most developed at this time because major work in the user application section would not begin until later in the project. Therefore, the initial estimate for the system software section was 1.4 million lines of code. After the revision, the new estimate was 1.88 million lines of code. That shows that there was a 34% size increase in this portion of the project. There was no planned system software development for increments 9 and 10.

The following table lists actual sizes for the first several increments as of February 1999. Note that no code was developed in increment one because it was primarily used to set up and test the processes and approach.
Table 21: Actual Increment Sizes

<table>
<thead>
<tr>
<th>Increment</th>
<th>Size (KSLOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>459.4</td>
</tr>
<tr>
<td>3</td>
<td>215.3</td>
</tr>
<tr>
<td>4</td>
<td>12.2</td>
</tr>
</tbody>
</table>

The size developed for the fourth increment was very small because more complex capabilities were deferred to later increments in order to keep the project on track. The project was already experiencing a potential drawback of following an evolutionary type process. The deferral of work to later increments can seem like a reasonable solution to a schedule problem, but it can also lead to an insurmountable amount of work being deferred until the very end of the project.

The actual data for the project at this point shows that the probability distributions used as input data for the model runs was appropriate for size, productivity, and defects. Productivity ranged between 1.6 LOC/Hr and 0.6 LOC/Hr. The higher productivity was seen for the first 3 increments. For these, all deliveries were internal without the scrutiny of full user acceptance into an operational environment. Therefore, it was possible to achieve many of the reuse goals which enabled the project to almost meet the original planned productivity of 1.8 LOC/Hr. The fourth increment had the first operational use of the system. There were unforeseen integration complexity, commercial product problems, and unplanned rework of code and that accounted for the lower productivity that was experienced later in the project. In addition, the project experienced a higher than expected turnover rate of 30%. These values are in agreement with the literature that states that productivity continues to fall in the range of 8-12 LOC/day (Jones
and that turnover for software development projects can average between 20 and 30% (Abdel-Hamid and Madnick 1991). The project was experiencing a size increase of approximately 40% and defect injection rates were higher than expected. At this point in the project, a small portion of the overall work had been accomplished, and yet management was already aware that the project could not be completed within the original budget and schedule. The simulation process model results demonstrate that this type of tool could have identified the tremendous risk that existed with the original budget and schedule for this project. By capturing the uncertainty that existed in early project size estimates and the dynamic and uncertain nature of productivity and defect injection rates, the models provide a means of assessing the impacts of unreliable information that exists when developing very early estimates. The outputs could have been used to provide interval estimates for effort and duration. The following table provides interval estimates for effort and duration based on information that was known at different points in the project:

Table 22: Interval Estimates at Different Points in the Project

<table>
<thead>
<tr>
<th>Date</th>
<th>Size Estimate (Million LOC)</th>
<th>Model Type</th>
<th>Effort Estimate (Labor Years)</th>
<th>Duration Estimate (Years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/96</td>
<td>1.4</td>
<td>Waterfall</td>
<td>[1466, 2214]</td>
<td>[6.1, 9]</td>
</tr>
<tr>
<td>9/96</td>
<td>3.425</td>
<td>Waterfall</td>
<td>[2123, 2190]</td>
<td>[8.9, 11.5]</td>
</tr>
<tr>
<td>9/96</td>
<td>3.425</td>
<td>Incremental</td>
<td>[2379, 2588]</td>
<td>[14.5, 16.7]</td>
</tr>
<tr>
<td>5/00</td>
<td>5.8</td>
<td>Waterfall</td>
<td>[2667, 3083]</td>
<td>[11.3, 13]</td>
</tr>
<tr>
<td>5/00</td>
<td>5.8</td>
<td>Incremental</td>
<td>[2887, 3061]</td>
<td>[15.8,17.8]</td>
</tr>
</tbody>
</table>
Two different software process simulation models were used to obtain the interval estimates. One model represents a typical waterfall process and the other model provides analysis for an incremental spiral type process. More can be learned about the process models in the article titled, “A Software Development Process Model of a Spiral Development Process” submitted to the ’06 Winter Simulation Conference. The first high level estimate for the project was completed at a time when the lifecycle approach had not been selected and when a very rough estimate of size was available. Based on this data, the waterfall process model shows that the schedule and effort estimate should have been higher than original budget of 1400 labor years and 5 calendar years since these values are not included in the interval estimates derived from the simulation model. The bottoms-up estimate was performed a short time later and an incremental, evolutionary lifecycle approach was considered. At this point, both types of process models could have been used to explore potential differences in cost and schedule. The results in Table 22 show that the spiral approach should take longer than a true theoretical waterfall approach because all requirements are considered fixed prior to beginning such a project. This is often an unrealistic goal for a project and leads to substantial cost and schedule impacts when requirements change due to the inflexible nature of the waterfall process. The project team decided to select the incremental spiral approach because past experience showed that a waterfall approach caused the project to expend a great deal of time and money on documents and requirements rather than development of the actual project. Even though this different approach was selected and the size estimate was greatly increased from the original 1.4 Million LOC, the budget and schedule did not change. Analysis with the simulation models could have identified that more funding and schedule were needed, especially if requirements were allowed to evolve through the different increments. Based on the uncertainty that existed prior to the project’s
start, the simulation models show that the project would most likely take longer than 8 years and would expend more than 2100 labor years of effort, no matter which lifecycle was followed. The project was re-planned once it became clear that the schedule was not feasible with deliveries every six months because the project was experiencing size growth of 72% and productivity rate of approximately 1 LOC/Hr. This actual project data is in agreement with the model inputs used for the simulation analysis.

The budget was raised to $400 million and time was added to the schedule so that the project would be completed in FY 06. Deliverables were scheduled for every 14 to 16 months. This budget and schedule is more in agreement with the results of the simulation model runs. Just before the project was cancelled, the estimate for the final product size was 5.8 Million LOC, with a total required effort of 2793 labor years and 10 years of schedule. Results from the simulation analysis that could have been done prior to project start are closer in agreement to these values than the original project estimate. The re-planning of the project included a substantial increase in funding and time, and yet it would prove to be inadequate, causing project managers to request additional funding.

Summary

This work has used simulation process models to study a NASA software development project that was plagued with funding and schedule problems. The use of simulation process models for analyzing effort and duration of a project based on uncertainty in key parameters of size, productivity, and defects was demonstrated for two different lifecycle approaches.
The CLCS project was given a budget and schedule that was approximately 50% of reasonable values. Extra funding and schedule would be added to the project, but the initial budget and schedule would have substantial impact on the project and would play a key role in it eventually being cancelled.

Cost and schedule estimation for large, complex software development projects is always challenging. None of the tools that exist today can provide an exact estimate and yet that is often what is expected by management. This project demonstrated how budgetary pressures can cause estimators to alter estimates to fit within the available budget. Even automated tools such as COCOMO rely heavily on qualitative and subjective input by experts. COCOMO is formula based, and yet a great deal of subjective input is involved in qualitatively rating attributes and selecting a suitable multiplying factor. Most cost estimation approaches do not account for uncertainty that exists when estimating before a project begins. Decision makers want to see an exact number, but need to understand that the complex dynamic interactions and process specifics that are not captured by this type of model but that have great impact, make it unlikely that a single number estimate will be exact. Simulation models provide a method of considering the development process and capturing uncertainty through the use of stochastic analysis with probability distributions for key factors. This work has demonstrated the benefits using a simulation process model to help project managers convey the complexity and uncertainty in the software development process. Hopefully, this can serve as a useful tool when debating unrealistically low budgets and schedules.

CLCS was in trouble from the very beginning and many close to the project knew that to be the case. Because of the initial budget and schedule, much time and effort was lost because of the need to defend why the project was late and why it should continue. Under-funding and too
short of a schedule had many negative and serious ramifications. An example is that there was a serious impact on retaining staff. Schedule pressure led to higher turnover rates and products were often delivered before they were ready in order to meet milestones and avoid cancellation. This led to a need to re-write a great deal of the code. The cost overruns and schedule slippages caused credibility damage with decision makers, especially when the overruns continued after a major re-planning of the project.
CHAPTER EIGHT: CONCLUSION

The complexity of software development has lead to the development and use of simulation models to evaluate the effects of the development process and the project environment for the commonly used waterfall process. This work has developed a simulation model of the spiral development lifecycle as well as an approach for using simulation for cost and schedule estimation. The goal is to provide a tool that can analyze the effects of a spiral development process as well as a tool that illustrates the difficulties management faces in forecasting budgets at the beginning of a project which may encourage more realistic approaches to budgetary planning.

Simulation is not typically used in the estimation process, with analytical cost estimation models and expert judgment techniques being the most commonly used by project managers for estimation. Simulation is especially valuable for the analysis of real-world systems that are too complex to allow for analytical evaluation. Since software development is inherently very complex, even the analytical cost models become complex, and can be misleading in giving managers a false hope of obtaining an accurate estimate with a mathematical solution. Studies show that no technique can provide an accurate estimate very early in a project’s life cycle, and yet early estimates are often cast in concrete and not allowed to be adjusted later in the project.

Analytic cost models such as COCOMO assume that the project environment is similar to that for which the model was developed or that adequate data exists to “tune” the model. Expert judgment techniques assume that experts’ memories are accurate and that pressure to adhere to a desirable budget can be ignored. Simulation can be used in conjunction with other tools to check the final outcome of different approaches and to test the validity of any assumptions.
Input probability distributions for size, productivity, and defect injection rates were added to the basic PATT model that was originally designed to accept single average point values based on data from the literature or industry. Lack of available data often leads to estimating parameter values for standard normal or triangular distributions rather than being able to fit a proper distribution for the data. Normal and lognormal probability distributions based on estimated parameters for task durations have been used in past models, but the use of fitted probability distributions for size, productivity, and defect injection rates is new (Kellnar 1991A; Raffo 1996). A large set of NASA data was used to fit acceptable distributions. Suggestions were made for distributions that could be used for environments where there is not adequate data. The use of the well documented Software Engineering Laboratory data for developing productivity and defect injection rate distributions is new. The parameters of the fitted distributions did fall in line with average values given in the literature, but the distributions provide a more complete representation of the values of productivity and defect injection rates that can be experienced in a similar environment. Size was included in the model as a uniform distribution based on sizing accuracy data found in Boehm’s *Software Cost Estimation with COCOMO II*.

The simulation model captures the key areas of uncertainty in size, productivity, and defect injection rates. The model also portrays the complexity of the process through graphical representation of the software moving through the various phases of the lifecycle. Confidence intervals for effort and duration can be calculated from many runs of the model. This provides a bounded check on effort and duration estimates derived using other techniques, and should cause concern if estimates derived using other techniques do not fall within the upper half of the confidence interval. Another benefit of using this approach is that it is more difficult to
manipulate the simulation to affect the outcome to a desired result. The use of probability
distributions and an established process flow make it more difficult to tweak individual values to
obtain a more popular estimate.

As a project progresses, the simulation model can be updated with actual data for
productivity, size, and defects and new estimates to complete can be derived using the same
methodology. In this way, the simulation approach emphasizes the uncertainty early in a project
and also allows analysis of actual project data. Estimation for a very large project should not be
a one time exercise. The Software Engineering Laboratory recommends a minimum of five re-
estimates, with the sixth estimate after system test being the only certain one (SEL 1990).

Trying to obtain a precise estimate at a very early stage in a project has lead to the use of
techniques that do not depict uncertainty and complexity of the factors. Human nature prefers a
single number for an estimate as opposed to a range of numbers, even though a range estimate
will have a much higher chance of including an accurate value (Boehm and Fairly 2000).
Simulation provides a technique that identifies risk and uncertainty based on the seemingly
random nature of the variables and the complexity of the project system.

The waterfall process provides for a structured sequential process, but this is often not
realistic since requirements tend to evolve and phases of the lifecycle may need to be repeated.
Because software development can be very complex and uncertain, iterative lifecycles are
gaining popularity. Therefore, a process model of an incremental spiral lifecycle was developed
so that the effects of such a process could be analyzed. This research shows that a spiral process
will most likely take longer than a true theoretical waterfall process because requirements evolve
and the process is repeated with additional process steps. In practice, a project that follows a
waterfall approach rarely captures all requirements perfectly up front and the lack of flexibility in
this process makes the impact of unknowns and requirements changes more severe than in the
more flexible spiral process that is designed to be very flexible. Therefore, either lifecycle
approach makes a one time estimate done at the beginning of a project very impractical. A rough
estimate can be developed, but detailed phased funding should be considered so that more
accurate estimates can be developed which are based on an incremental basis rather than for the
entire project. The intended benefit of spending more time and money on such a process is that
the user will get a better product and will get incremental functionality with each delivery. This
work also shows that uncertainty in areas such as size, productivity, and defects should be
accounted for when developing an estimate, no matter which lifecycle is selected. Size growth
due to requirements changes and unknowns can be extensive. The literature points out that very
early size estimates are likely to be much lower than the actual final size of the project due to
requirements changes and unknowns (SEL 1994; Boehm, Abts et al. 2000). This is often the case
with either the waterfall or spiral process, even though requirements should not drastically
change in a true theoretical waterfall lifecycle. Requirements evolution is a key part of a spiral
process and therefore, plans for accommodating size changes should be considered when
estimating a project that will follow such a process. This is a more realistic view of the situation
for large, complex projects than a theoretical waterfall process.

The spiral process model can be used throughout a project to analyze the project as more
information becomes available. For instance, data from early increments can serve as inputs to
the model and an estimate to complete based on this data can be assessed.

Process simulation models and system dynamic models each have strengths for analysis
of software development projects, so it is desirable to combine the two in order to enable more
thorough analysis of the process and environment. This research has demonstrated how a user-
friendly system dynamics tool can be combined with a discrete event process model for analysis of the effects of turnover on cost and schedule. Turnover is a common problem for software development organizations and its potential impact on the project needs to be considered when planning for the project. The literature estimates that the average turnover rate is approximately 30% per year and this research has shown that this level of turnover will cause effort and duration estimates to increase for a project. The method of combining the two models for this work provides a tool that readily emphasizes why such a factor should be considered and how to best prepare for the possible impacts.

Projects that face budgets and schedule that are 50% less than what is reasonable are considered death march projects that have no real chance of success. Unfortunately, this is not a rare occurrence for software development projects, and yet this situation can have substantial and highly undesirable impacts. The NASA case study that was used for this research identified some of the negative impacts that the very low budget and schedule had on the project. These included loss of team morale, high turnover, lower productivity, greater amount of rework, and loss of credibility with decision makers. A major re-planning of the project would provide more funding and schedule, but the project was never able to fully recover and was cancelled prior to its completion. This raises the question of whether or not the project would have been successful had it been given a more realistic budget and schedule from the beginning.

Suggestions for Future Work

This work has demonstrated the potential benefits of using simulation when developing cost and schedule estimates by considering the lifecycle process used, the uncertainty in key
parameters, and the effects of turnover. It is desirable to further test and validate the cost estimation simulation approach developed for this work. The NASA case study provided several data sets for analysis, however, additional data from multiple projects in the NASA environment and other environments would serve to further assess the benefits of such an approach.

The probability distributions for this work were based on subsets of data from the Software Engineering Laboratory because the original servers that contain the full data sets are no longer maintained. There is a possibility that access ability to the entire data set may be restored, and if this occurs, it would be interesting to fit the entire data set to probability distributions. There is also a need to analyze more recent project data in the NASA environment, especially for projects that have followed newer lifecycle approaches. Further, it would be interesting to consider the development of probability distributions for additional parameters of interest such as defect detection and correction rates.

More work can be done to refine the spiral development process model. The collection and analysis of additional data from other projects that have followed a spiral development lifecycle would be useful for refining the model and for additional analysis.

The combination of the system dynamics model and the discrete event process model demonstrated a unique approach to combining the two types of simulation models. The system dynamics model was a very simplified human resource model and more detail could be added to the model to further analyze human resource issues and effects. In addition, it would be interesting to alter the discrete event process model to include continuously changing resource pools within the model.
Conclusion

This research has demonstrated the benefits of using a simulation model for cost and schedule estimation to provide more realistic results including an interval estimate to help focus on the uncertainty of the estimates. Data from a past NASA project that experienced cost and schedule problems was used for this work. Changes were made to a discrete event process simulation model to include size, productivity and defect injection probability distributions. Comparison of the estimates calculated using the simulation approach demonstrated that the cost and schedule were about 40-50% less than what was reasonable for a budget. It is the hope of this research that future projects that face an unrealistic budget will find the simulation approach helpful for debating estimates that do not adequately consider the complexity and uncertainty of software development and its effects on cost and schedule estimation.
APPENDIX PERMISSION FOR USE OF COPYRIGHT MATERIAL
Carolyn Mizell  
1672 Shore Drive  
Merritt Island, FL 32952  
February 16, 2006  

IEEE Intellectual Property Rights Office  
445 Hoes Lane  
Piscataway, NJ 08855-1331  

Dear Intellectual Property Rights Office:

I am completing a doctoral dissertation at the University of Central Florida entitled “Quantitative Assessment of Software Development Project Management Issues Using Process Simulation Modeling with Continuous Simulation Elements”. I would like your permission to reprint in my dissertation excerpts from the following:


The excerpt to be reproduced is Figure 2: Spiral model of the software process on page 64.

The requested permission extends to any future revisions and editions of my dissertation, including non-exclusive world rights in all languages, and to the publication of my dissertation by UMI. These rights will in no way restrict publication of the material in any other form by you or by others authorized by you. Your signing of this letter will also confirm that your company owns the copyright to the above-described material.

If these arrangements meet with your approval, please sign this letter where indicated below and return it to me in the enclosed return envelope. Thank you for your attention in this matter.

Sincerely,

Carolyn Mizell

PERMISSION GRANTED FOR THE USE REQUESTED ABOVE:

By:  
Date: 2/23/06
MAR 7, 2006

CAROLYN MIZELL
1672 Shore Drive
Merritt Island, FL 32952

Dear Ms. Mizell:

You have our permission to include content from our text, SOFTWARE COST ESTIMATION WITH COCOMO 2, 1st Ed. by BOEHM, BARRY W.; ABTS, CHRIS; BROWN, A. WINSOR; CHULANI, SUNITA; CLARK, BRADFORD K.; HOROWITZ, ELLIS; MADACHY, RAY; REIFER, DONALD; STEECE, BERT, in your dissertation at the University of Central Florida.

Content to be included is:
p. 10 Fig. 1.2

Please credit our material as follows:


Sincerely,

Michelle Johnson
Permissions Administrator


Basili, V. R., F. E. McGarry, et al. (2002). Lessons learned from 25 years of process improvement: The Rise and Fall of the NASA Software Engineering Laboratory. ICSE, Orlando, FL.


