THE ADVANTAGES OF IMPLEMENTING SOFTWARE ENGINEERING PROCESS MODELS

by

RICKY DON PREUNINGER

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2006

ACKNOWLEDGEMENTS

I would like to offer my thanks and appreciation to all the people who have provided their support in my effort to achieve my degree. First, I want to thank my wife, Glynna, who always provided me with her encouragement and devotion especially when I was discouraged; my children, Donald and Maria, who understood the sacrifice of time with them. I would like to thank my managers who encouraged me through the years, Robert M. Hensell (1996-1998); Ronald L. Peck (1998-2002); director, Phillip Kohlruss (1998-to date); L. Rodney Barthold (1985-1996), who urged me to start my masters degree; and especially the late Elizabeth B. Jones (2002-2005), who urged me not to give up and was a constant source of advise. I appreciate the generous help, support and counsel that my advisor, David Levine, gave to me in completing this thesis.

April 10, 2006

ABSTRACT

THE ADVANTAGES OF IMPLEMENTING

SOFTWARE ENGINEERING

PROCESS MODELS

Publication No. _____

Ricky Don Preuninger, M.S.

The University of Texas at Arlington, 2006

Supervising Professor: David Levine

The North Atlantic Treaty Organization Science Committee had discussions on the topic concerned the state of Computer Science. There were worldwide issues with the development of software, the crisis being that software projects did not seem ever to complete. The study group coined the term “software engineering” to be provocative and implying need for software manufacturing to be similar to traditional branches of engineering. In the beginning, individual programmers used whatever means worked to build software. Formal methods of design or programming did not exist. Programmers were never able to give a definitive estimate as to how long a project would take. Software projects often were behind schedule, over cost, poorly documented, poorly

designed, and contained items other than the requirements. These projects were costing corporations thousands of dollars. The software industry was quite undisciplined and it was obvious.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF EQUATIONS

CHAPTER 1

INTRODUCTION SOFTWARE AND SOFTWARE ENGINEERING

The North Atlantic Treaty Organization (NATO) Science Committee started discussions in early 1967; the topic concerned the state of Computer Science [NAU69]. There were worldwide issues with the development of software, the crisis being that software projects, for the most part, did not seem ever to complete. In late 1967, the study group coined the term "software engineering" to be provocative and implying the need for software manufacturing to be similar to the traditional branches of engineering. A conference held in 1968 introduced the new concepts to the attendees in Garmisch, Germany [NAU69].

Since the NATO conference, software engineering is still having problems with software projects being behind schedule and over budget, with many efforts introduced in an effort to correct those problems.

1.1 Software Engineering Terms and Definitions

1.1.1 Software

Software is a methodical set of instructions (computer program) for execution on hardware (a computer) to provide the desired features, function, and performance [PRE04]. Software can adequately manipulate information in data structures. The manufacturing methods of software and hardware are different. Software is more of an

engineering or development type process and unlike traditional systems; software does not wear out [PRE04].

1.1.2 Software Engineering

Software Engineering is a systematic approach of applying sound engineering principles in order to obtain software, which will run reliably and effectively on a computer [PRE04]. Software engineering applies various stages to implement all of the components needed to satisfy the requirements [NAU69]. Software engineering follows traditional engineering which is "to contrive or plan out, usually with more or less subtle skill and craft" or "to guide the course of" [MER01].

1.1.3 Computer Science

Computer science is the methodical study of computing systems and computation. The organization of knowledge resulting from this order contains theories for understanding computing systems, design methodology, algorithms, the testing methods of concepts; methods of analysis and verification [NAU69].

1.1.4 Design

Design is "to create, fashion, execute, or construct according to plan" or "to conceive and plan out in the mind" [MER01]. Design is the effort, which the engineer performs to create an answer to satisfy the customer requirements or problems.

1.1.5 Programmer

Programmers are individuals who take the software design which the software engineers have developed and produce the instructions (software code) for the

computer. The instructions are the components, which make up the software. The computer is then able to execute those instructions or software [PRE04, MER01].

1.1.6 Formal Method

A method is formal when determined to have a sound mathematical basis when given by a formal specification. These are the basis for providing precisely the necessary components so that the software is consistent and complete [PRE04].

1.1.7 Estimating

Estimating is the process of determining how much the product will cost to design, produce, and test a piece of software that will satisfy all of the given requirements. Estimating is used to describe the amount of time and money needed to accomplish the design, production and testing of the software [MER01].

1.1.8 Projects

Projects are tasks or problems engaged in usually by a team or teams. Projects will include all of the elements, which the customer has requested or ordered. The completion of these elements can determine when the customer will make payments.

1.1.9 Project Management

Project Management is the act or art of managing the individual projects [MER01]. Effective management can result in a successful project. It is questionable whether Project Management applies across all corporations consistently and generically. Project management is typically responsible for all aspects of the project [CRA06]. This effort includes planning, monitoring, control of the people assigned to the project (engineering staff, programmers, estimators, cost/schedule planners, etc.),

control of the processes and control of the events as the software evolves from a conceptual phase to an operational phase [PRE04].

1.1.10 Processes

Processes are systematic and disciplined steps taken which lead to the completion of the projects. The software process is the outline for the tasks that are necessary to design, produce, and test high-quality software [PRE04].

1.1.11 Process Management

Process Management is controlling the processes used in developing software. Management of software engineering processes has a significant bearing on the quality of the software product [LI02]. "The software process is characterized by the complexity of both the product and the process and the dynamically changing environment" [JOE97].

1.1.12 Cost Drivers

Cost Drivers are different scalars or effort multipliers used in some of the software cost estimating tools. These drivers are broken up into four categories; product, system platform, personnel and project factors [BOE95].

1.1.13 Software Configuration Management (SCM)

Software Configuration Management (SCM) is a management discipline to control the software application development. SCM is to assist project management to ensure product integrity and control software changes in an orderly method that will satisfy the customer requirements.

1.2 Software Engineering

The need for Software Engineering came soon after the invention of computers. In the beginning, the individual programmer used whatever means worked to build software. Formal methods of design or programming did not exist. The programmer was never able to give a definitive estimate as to how much time a project would take or the amount of resources needed to complete the project. Management had no way of accurately estimating the cost of a new project. The software projects often were behind schedule, over cost, poorly documented, poorly designed, and contained items other than the requirements [BER78, PAU93]. These projects were costing corporations thousands of dollars [HU98]. The software industry was quite undisciplined and it was obvious [BER78]. One management view was that "I would rather have the project wrong than to have it late" [HAR00].

1.3 Project Impacts

A 1984 survey revealed that from a sample of twenty-three corporations with seventy-two software projects, cost overruns were an average of 67 percent and schedule slippages about 22 percent [HU98]. A 1987 survey revealed that the average cost overruns per project were approximately 225,000 dollars with a three-month schedule slippage [HU98]. A study performed by Peat Marwick Mitchell and Co. revealed over 35 percent of their 600 largest customers had major software project cost overruns and schedule slippages [HU98].

1.4 Software Engineering Process Management

Software Engineering Process Management evolved out of the need to manage the increased size and complexity of software projects. Process management came about because of the belief that the software processes could be analyzed, designed and maintained just as if it were a piece of software [ZHA05]. This view lends itself to software process modeling (SPM) [ZHA05].

CHAPTER 2

SOFTWARE ENGINEERING BACKGROUND

Software engineering has evolved a great deal since that first conference, held in October 1968. The software industry has grown rapidly over the last several decades. Projects' software cost ratio increased from 3:7 software to hardware in 1980 to 1:1 in 2003 [LI02, BER03]. One study showed that the Information Technology (IT) projects have grown by 500 percent from 1990 to 2000 [HAR00]. The United States (US) Department of Defense (DoD) found that their largest cost impacts on programs were software with 1.5 million computers, 28,000 systems, and 10,000 networks [GAO01]. Government and industry has seen large cost impacts with project software cost growth of this size.

2.1 Cost Impacts

In 1997, the estimated cost of software projects to corporations and governments was millions of dollars. The software development process has always been a challenge of software engineering and needed management controls in place to regulate the cost and schedule of these projects [JOE97].

Several attempts made over the years to control project cost and schedule resulted in the building of many models [HU98]. A few of the attempts were implementation of Cost/Schedule Control System Criteria (C/SCSC) [CHA94, PRE03], Software Configuration Management (SCM) [BER78, JOE97, REN91], Earned Value

Management Systems (EVMS) [PRE03] and International Standards Organization (ISO) 9001 [BIB02].

2.1.1 Cost/Schedule Control System Criteria (C/SCSC)

In the 1970's, the US DoD introduced the Cost/Schedule Control System Criteria (C/SCSC) in their DoD 7000.2 document [CHA94, PRE03]. This was due to the rising cost of all projects in an effort to control the cost and schedule of the projects. The intent of C/SCSC was really metrics, to measure the condition of a project to determine how well the project management controls were in place. These metrics were not effective with the software engineering processes.

The C/SCSC was using formulas to determine the health of the program. The formulas were for Cost Variance (CV) (Equation 2.1), Schedule Variance (SV) (Equation 2.2), Cost Performance Index (CPI) (Equation 2.3), Estimate at Completion (EAC) (Equation 2.4) Schedule Performance Index (SPI_1) (Equation 2.5) and Total Cost Performance Index (TCPI) (Equation 2.6) [CHA94]. Figure 2.1 shows how the cumulative cost data will typically appear on a project.

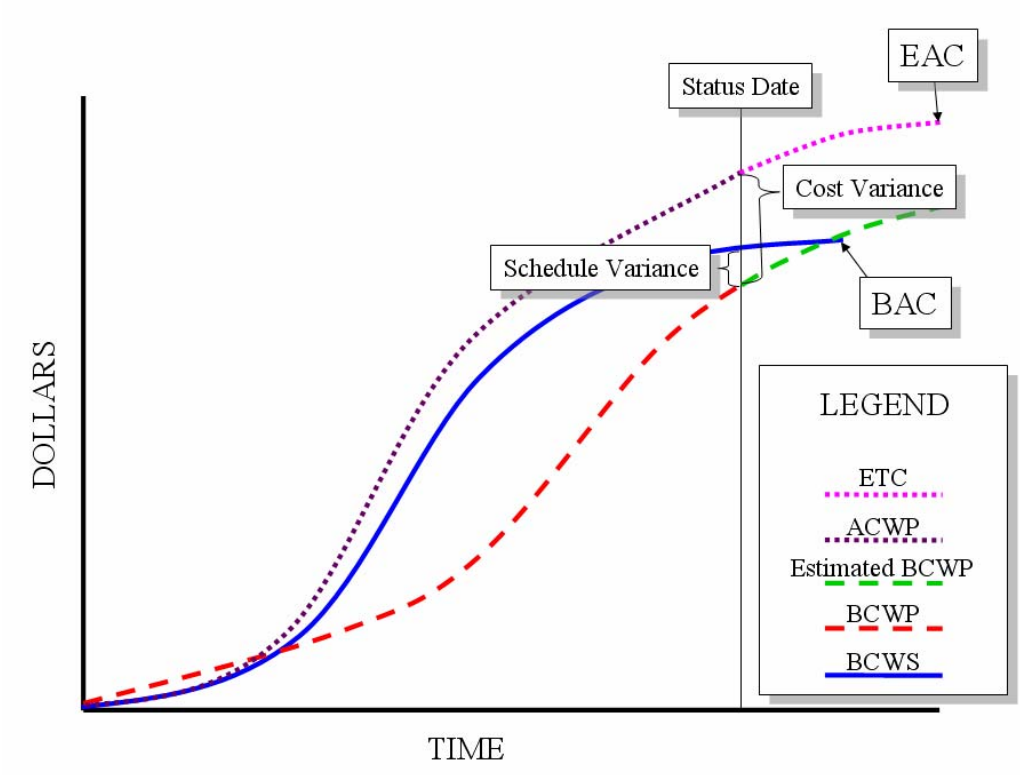

Figure 2.1 Cost Schedule Chart [PRE04]

BCWS is Budget Cost of Work Scheduled; BCWP is Budget Cost of Work Performed; ACWP is Actual Cost of Work Performed; ETC is Estimate to Complete; BAC is Budget at Complete; and EAC is Estimate at Complete.

Project management and DoD determined the health of the project using metrics in the figure 2.1, Equations 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, and 2.9. The chart shows from a cost standpoint how much the projects are under/over spent and head/behind schedule. The figure 2.1 gives a graphical view while the equations provide a numerical value of the project health.

Equation 2.1 Cost Variance (CV) [CHA94]

$$CV = BCWP - ACWP$$

$$CV\% = \frac{BCWP - ACWP}{BCWP} \times 100$$

$$CV = 180 - 225 = -45$$

$$CV\% = \frac{180 - 225}{180} \times 100 = -25\%$$

Equation 2.1 works with either dollars or hours. So if a project has completed work which was budgeted to be done in 180 dollars and actually took 225 dollars to complete then the cost variance is -45 dollars or -25 percent. This would indicate that the project has overspent 45 dollars on performing the completed tasks or the project is costing 25 percent more than expected. This would inform project management that for every dollar of work performed, the project is costing an extra 25 cents [LM04].

Equation 2.2 Schedule Variance (SV) [CHA94]

$$SV = BCWP - BCWS$$

$$SV\% = \frac{BCWP - BCWS}{BCWS} \times 100$$

$$SV = 180 - 250 = -70$$

$$SV\% = \frac{180 - 250}{250} \times 100 = -28\%$$

In Equation 2.2, if a project has completed work which was budgeted to be done in 180 dollars and was scheduled to have completed 250 dollars then the schedule variance would be -70 dollars or -28 percent. This would indicate that the project is behind schedule. The project had planned to accomplish 70 dollars more work than accomplished or is 28 percent behind schedule. This would inform project management that tasks in the project are taking approximately 3 days extra for each 10 days of work [LM04].

Equation 2.3 Cost Performance Index (CPI) [CHA94]

$$CPI = \frac{BCWP}{ACWP} \times 100$$

$$CPI = \frac{180}{225} \times 100 = 80\%$$

CPI is the cost performance index, which is the cost efficiency of the project. CPI below 1.00 is unfavorable and above 1.00 is favorable. The project is performing well if the CPI remains above 95, advisory condition when between 90 and 95, and critical condition when below 90. With the BCWP and ACWP at 180 dollars and 225 dollars respectively then the project's CPI is at 80 percent. This index states that for every dollar that is spent on the project the customer gets an 80-cent value for each one dollar spent [LM04].

Equation 2.4 Estimate at Completion (EAC) [CHA94, LM04]

$$EAC = ACWP \times \frac{BAC - BCWP}{CPI}$$

Or

$$EAC = \frac{BAC}{CPI}$$

$$EAC = 225 + \frac{300 - 180}{.80} = 375$$

$$EAC = \frac{300}{.80} = 375$$

Equation 2.4, the CPI will give project management and customer the ability to make a prediction for the EAC that if the BAC is 300 dollars then the EAC will likely be 375 dollars. This provides an indicator to the customer and upper management if the reported EAC by project management or individual departments is understated. The project manager might be understating the EAC in order to protect the project from cancellation. If the EAC reported on the project is 325 dollars then there is a 50-dollar disagreement. If this disagreement is significant then there should be a review of the project.

Equation 2.5 Schedule Performance Index (SPI_1) [CHA94]

$$SPI = \frac{BCWP}{BCWS} \times 100$$

$$SPI = \frac{180}{250} \times 100 = 72\%$$

Equation 2.5, when the SPI_1 is below 1.00 then it is unfavorable and above 1.00, it is favorable. The project is performing well if the SPI remains above 95, advisory condition when between 90 and 95, and critical condition when below 90. The BCWP and BCWS are 180 dollars and 250 dollars respectively then the project's SPI is 72 percent. This would indicate to project management an accomplishment of only 72 percent of the planned work [LM04].

Equation 2.6 Total Cost Performance Index (TCPI) [CHA94]

$$TCPI = \frac{BAC - \sum BCWP}{EAC - \sum ACWP} \times 100$$

$$TCPI = \frac{300 - 180}{325 - 225} \times 100 = \frac{120}{100} \times 100 = 120\%$$

TCPI is the total schedule performance index. Maintaining this efficiency is necessary to achieve the budgetary goal. TCPI above 1.00 is unfavorable and below 1.00 is favorable. The BAC is 300; the EAC is 325; cumulative BCWP are 180;

cumulative ACWP are 225; and the project's TCPI is 120 percent (as shown in equation 2.6). This indicator states that based on the current EAC, 1.20 dollars worth of work is necessary to accomplish each dollar of cost expected to complete the tasks [LM04].

Equation 2.7 Variance at Completion (VAC) [LM04]

$$VAC = BAC - EAC$$

$$VAC = 300 - 325 = -25$$

VAC is the variance at completion of the project. BAC and EAC are 300 and 325 dollars respectively then the VAC is -25 (as shown in equation 2.7). This equation is predicting that the project will overrun by 25 dollars. Using the EAC of 375 from Equation 2.4, the VAC is -75 dollars, which might be a concern for upper management, and the customer [LM04].

Equation 2.8 Percent Complete (PC) [LM04]

$$PC = \frac{\sum BCWP}{BAC} \times 100$$

$$PC = \frac{180}{300} \times 100 = 60\%$$

PC is the percent complete of the project based on the cumulative BCWP and the BAC, which are 180 and 300 respectively. The percent complete is 60 percent (as shown in Equation 2.8). This provides project management and the customer with an approximate status of the project [LM04].

Equation 2.9 Percent Spent (PS) [LM04]

$$PS = \frac{\sum ACWP}{BAC} \times 100$$

$$PS = \frac{225}{300} \times 100 = 75\%$$

PS is the percent spent of the project budget based on the cumulative ACWP and the BAC, which are 225 and 300 respectively then the percent spent is 75 percent (as shown in Equation 2.9). This provides project management and the customer with an estimate that the project has spent 75 percent of the budget while the project is only 60 percent complete [LM04].

2.1.1.1 Advantages of C/SCSC

The main advantage of C/SCSC is providing the customer and management with a quick overview of the project status. C/SCSC gives management the ability to see when a program is getting into trouble prior to the program actually being in trouble.

2.1.1.2 Disadvantages of C/SCSC

The main disadvantage of C/SCSC is that it is not a complete picture of the project and is only a metric. The problem areas need reviewing to see what the cause of the problem is and decide what the corrective action should be on the project. The schedule status is in a dollar figure rather than a time scale [LIP03].

2.1.2 Software Configuration Management (SCM)

Introduction of Software Configuration Management (SCM) came about because software was relatively easy to change. This created the issue that since software was easy to change there was always a new change being introduced resulting in the projects not finishing [BER78, BER03]. The intent of SCM was to introduce a method of tracking code changes within the software processes [BER78, BER03, HAR00, and JOE97]. SCM intended to provide discipline similar to the configuration management used in other manufacturing projects [BER78, BER03]. The belief was that the discipline of SCM would to be able to resolve the traditional software development problems such as cost overruns, schedule delays, and unsatisfied customer requirements. The discipline would provide management control over what changes with approvals prior to implementing them into the product [BER78, BER03]. The way to perform this was to have the software broken up into Software Configuration Items (SCI) and change management on these SCIs. A key factor of the software development and maintenance process is SCM [JOE97, BER78]. Code change management was one of the core problems of software development [JOE97].

There are two views of SCM; one being a technical view and the other a management view. While there were similarities, there is a definite gap between the technical and management views [JOE97].

2.1.2.1 Technical View of SCM

The technical view of SCM is to provide tools to change processes by implementing change management methods. Doing this handles the change requests and performs changes in a controlled manner [JOE97, BER78, REN91, and BER03]. The version control of software provides consistent software configuration and is the major part of SCM [JOE97, BER78, BER03, and HAR00]. This could be a very detailed and comprehensive process support to the SCM [JOE97].

2.1.2.2 Management View of SCM

The management view focuses on the organizational and administrative aspects [JOE97]. The change management process handles the change requests and performs the changes within a controlled manner with controlled processes [JOE97, BER78, REN91, and BER03]. This process is often a very course informal process model [JOE97].

2.1.2.3 Advantage of SCM

The advantage of SCM is the control provided in managing changes made to the software product. The objective of SCM is the cost-effective management of a system's life cycle [BER78].

2.1.2.4 Disadvantage of SCM

The major disadvantage of SCM is the fact that SCM is relatively immature compared to the hardware Configuration Management (CM). One of the major issues, which cause SCM to fail, is trying not to implement SCM like hardware CM [BER78].

2.1.3 Earned Value Management System (EVMS)

Electronic Industries Alliance (EIA) published EVMS as a replacement for C/SCSC and has become an ANSI standard (ANSI/EIA 748-98). DoD did not formally accept the change until December 1996 [DOD96, PRE03]. The creation of EVMS reflected that there was more than cost and schedule needing control in order to manage the project. Like C/SCSC, EVMS is metrics to show how well project management is working. The formulas used are the same formulas used in C/SCSC.

2.1.3.1 Advantages of EVMS

The main advantage of EVMS is the same as with C/SCSC. EVMS gives the customer and management with a quick overview of the project status. EVMS gives management the ability to see when a program is in trouble.

2.1.3.2 Disadvantages of EVMS

The main disadvantage of EVMS is the same as with C/SCSC. EVMS is not a complete picture of the project and is only a metric. The problem areas need reviewing to see what the cause of the problem is and decide what the corrective action should be on the project. The schedule status is in a dollar figure rather than a time scale [LIP03].

2.1.4 International Standards Organization (ISO) 9001

The ISO 9001, developed by the International Standards Organization, is a series of standards certifying the quality management and assurance of an organization [BIB02, PRE04]. One of the standards, the ISO 9000-3, certifies the software development organizations. ISO 9001:2000 is a generic standard for organizations who want to improve the overall quality of their products [PRE04].

2.1.4.1 Advantages of ISO 9001

The advantages of ISO 9001 are the framework, which contains well-documented procedures, high quality standards, consistent checkpoints, and good communication with subject matter experts [HYS99]. ISO 9001 provides discipline to the development of the products.

2.1.4.2 Disadvantages of ISO 9001

The main disadvantage of ISO 9001 is that ISO 9001 is not always easy to implement and is very expensive. ISO 9001 is very difficult for a small company to implement and may leave them where they are not competitive [HYS99]. ISO 9001 only implements the quality control procedures and not the training of quality itself.

2.1.5 Software Cost Estimating Models

There are cost and schedule models developed purely for use on software engineering projects to estimate the cost and schedule. A couple of the models are the COnstructive COst MOdel (COCOMO) and the Software Life Cycle Model (SLIM) [HU98]. These models require that and organization know the processes used so that past project experiences can be used to estimate the future projects. In many cases, the

needed data is not being collected or a model is not used [HU98]. The cost models must use relevant cost data to provide usable information. Surveys indicated that some projects were using 1960s and 1970s cost drivers and the relevancy is unknown to today's cost drivers [HU98]. These models do not perform well if the models are not calibrated to the project [HU98]. Another key to using the software cost models is to know the variables in the model and how they were developed to provide a more meaningful estimate [HU98].

2.1.5.1 COCOMO

COCOMO is one of the earlier software cost models. Barry Boehm developed COCOMO and is one of the more widely accepted cost models used today [HU98]. COCOMO is the benchmark, which many other cost models have been measured to over the years [HU98]. The model has evolved into a more complex and comprehensive model. COCOMO II model is the name of the new improved model [PRE04]. The model has three submodels based on the complexity of the project: Basic, Intermediate, and Detailed. Equation 2.10 is an example of the formula.

Equation 2.10 COCOMO II [BOE95]

$$E = 2.94 \times EAF \times (S)^{C}$$

$$E = 2.94 \times 1.0 \times (9)^{1.10}$$

$$E = 2.94 \times 1.0 \times 11.21$$

$$E = 32.96 \text{ Person-months}$$

Equation 2.10 will use the several elements. *EAF* is Effort Adjustment Factor derived from the cost drivers. *S* is the software size measured in thousand lines of code (KLOC). *C* is five scale drivers, which are Precedentedness, Development Flexibility, Architecture / Risk Resolution, Team Cohesion and Process Maturity [BOE95]. Therefore, if the project had nominal cost and scale drivers then *EAF* could be 1.0 and *C* could be 1.10. If the project has roughly 9,000 lines of code then the equation would look like Equation 2.10.

The teams developing a cost estimate of a new project normally uses the COCOMO model. The model uses data from prior projects to project the new estimate. If prior project data is not available, the model will still provide an estimate but will not be as accurate as it would be if the data existed.

2.1.5.2 SLIM

SLIM was developed by L. H. Putnam in 1978 and was revised in 1992. The SLIM equation is as shown in the Equation 2.11 [HU98] or as shown in equation 2.12. The two Equations are the same solved for different variables. Equation 2.11 solves for the source lines of code and Equation 2.12 solves for person-years.

Equation 2.11 SLIM [HU98]

$$S = P \times \sqrt[3]{\left(\frac{E}{B}\right)} \times t^{\frac{4}{3}}$$

$$S = 2{,}000 \times \sqrt[3]{\left(\frac{2}{1.05}\right)} \times 3^{\frac{4}{3}}$$

$$S = 2{,}000 \times 1.24 \times 4.33 = 10{,}738.4 \text{ SLOC}$$

Equation 2.12 SLIM [HU98]

$$E = \left(\frac{S \times \sqrt[3]{B}}{P}\right)^3 \times \frac{1}{t^4}$$

$$E = \left(\frac{9{,}000 \times \sqrt[3]{1.05}}{2{,}000}\right)^3 \times \frac{1}{3^4}$$

$$E = \left(\frac{9{,}000 \times 1.016}{2{,}000}\right)^3 \times \frac{1}{81}$$

$$E = (4.572)^3 \times 0.0123$$

$$E = 95.57 \times 0.0123 = 1.176 \text{ Person-years}$$

Equation 2.11 and 2.12 will use the several elements. The *S* is source lines of code (SLOC). *P* is a production parameter. *E* is the effort in person-years. *B* is the skill factor and *t* is the total development time in years or months [HU98]. If *P* is 2,000,

B is 1.05 and *t* is 3 years. For Equation 2.11, *S* is 9,000 or for Equation 2.12, *E* is two person-years.

The teams developing a cost estimate of a new project also uses the SLIM model. The model uses data from prior projects to project the new estimate. The model does not perform as well if the prior project data is not available.

2.1.5.3 Advantages of Software Cost Estimating Models

The main advantage of the Software Cost Estimating Models is that they provide a basis for estimating new project cost.

2.1.5.4 Disadvantages of Software Cost Estimating Models

The main disadvantage of the Software Cost Estimating Models is that they need prior project data to perform well on projecting new projects.

2.1.6 Software Engineering Process Models

The development of software engineering process models was to provide a guide for the development of software applications. The models have evolved and improved over the years. The four basic models are Waterfall, Spiral, Incremental, and Agile.

2.1.6.1 Waterfall

The waterfall process model is where one task is finished after the other. This model is the classic life cycle because of its methodical chronological flow [PRE04]. This process was one of the first to be developed. When Winston Royce proposed this model, he made provisions for feed back loops, but for simplicity basis, the

organizations implementing this model commonly leave them out. The steps are requirements, planning, design, code, integration testing, and delivery [PRE04].

2.1.6.1.1 Waterfall Process Steps

Figure 2.2 leaves out the feed back loops for simplicity. Anytime during the planning step the organization can take the feed back loop into the requirements step to clarify or obtain more needed requirements.

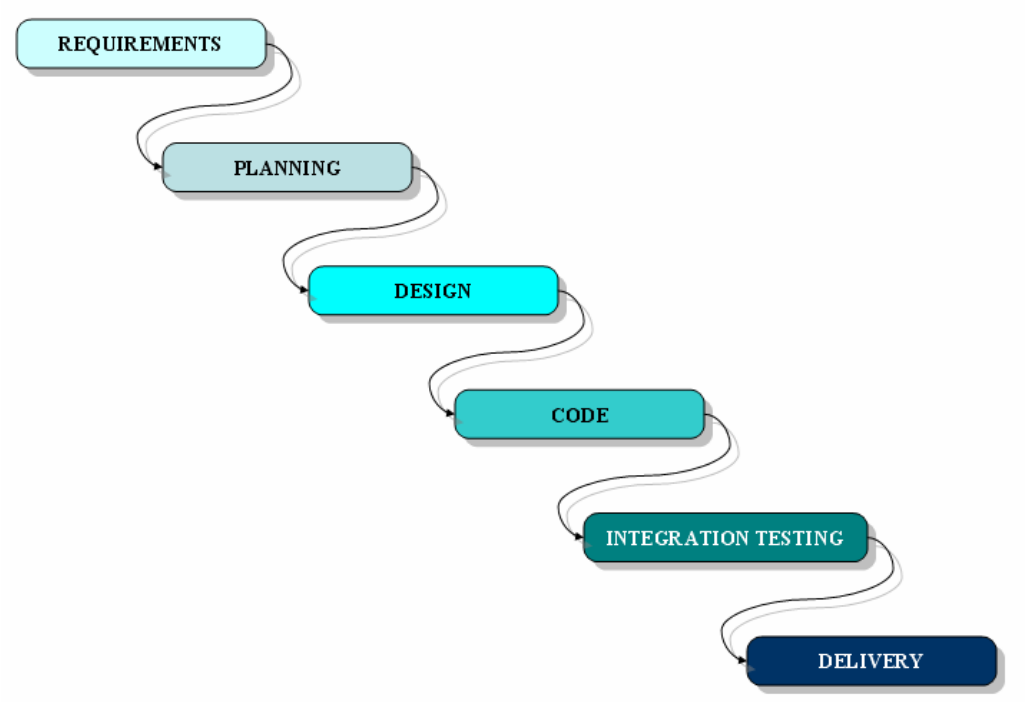

Figure 2.2 Waterfall [PRE04]

2.1.6.1.1.1 Requirements

The requirements step is the collecting of all the customer requested requirements of the software application. These requirements can come in many forms and documents. They are in the contract, provided through requirement documents,

email or even by a verbal telephone call. The more formally provide requirements are less likely of changes or misunderstandings. The requirements should provide everything that the software is to accomplish and should state if there is expected response time [PRE04].

2.1.6.1.1.2 Planning

The planning step is going through all requirements and determining what steps to accomplish the requirements. This is the time when the organization has the ability to determine what the cost and schedule is for the project [PRE04].

2.1.6.1.1.3 Design

The design step is when the software engineering takes elements in the requirements, analyzes the elements, and lays out the plan of the application to fulfill all of those elements. The plans of the application will layout all of the required code modules [PRE04].

2.1.6.1.1.4 Code

The code step is taking the design and building code for modules. During this step, each module goes through unit testing to verify that code is performing as designed and is fulfilling its portions of the requirements [PRE04].

2.1.6.1.1.5 Integration Testing

The integration step is taking all of the modules and compiling them together into a single application for testing. During this testing phase, verifies that code is performing as designed and fulfilling all customer requirements [PRE04].

2.1.6.1.1.6 Delivery

The delivery step is the delivery of the application to the customer who requested the product. Delivery also includes the support and feedback to the customer [PRE04]. This step can be one of the longest steps depending on how long the product is in service.

2.1.6.1.2 Advantages of Waterfall

The basic advantage of the waterfall model is the straight systematic flow of the processes, which are easily understood.

2.1.6.1.3 Disadvantages of Waterfall

One of the main disadvantages of the waterfall model is that the customer does not see what the product looks like until delivery, so the customer must be very patient until then. At the time of delivery, the team is not certain that the customer's requirements are satisfied. The customer seldom states all of the requirements at the beginning of the project. If the customer interjects with very many requirement changes, the model becomes somewhat confusing to the development team [PRE04]. This model does not take into account that the product may need additions and maintenance.

2.1.6.2 Spiral

The spiral process model has the same steps as the waterfall process model. Development of the spiral process model was due to realizing that requirements given at project start are incomplete; requirements and design are continuously evolving as time

passes [PRE04]. The steps go through a cyclical motion as requirements and design evolve as shown in figure 3.2.

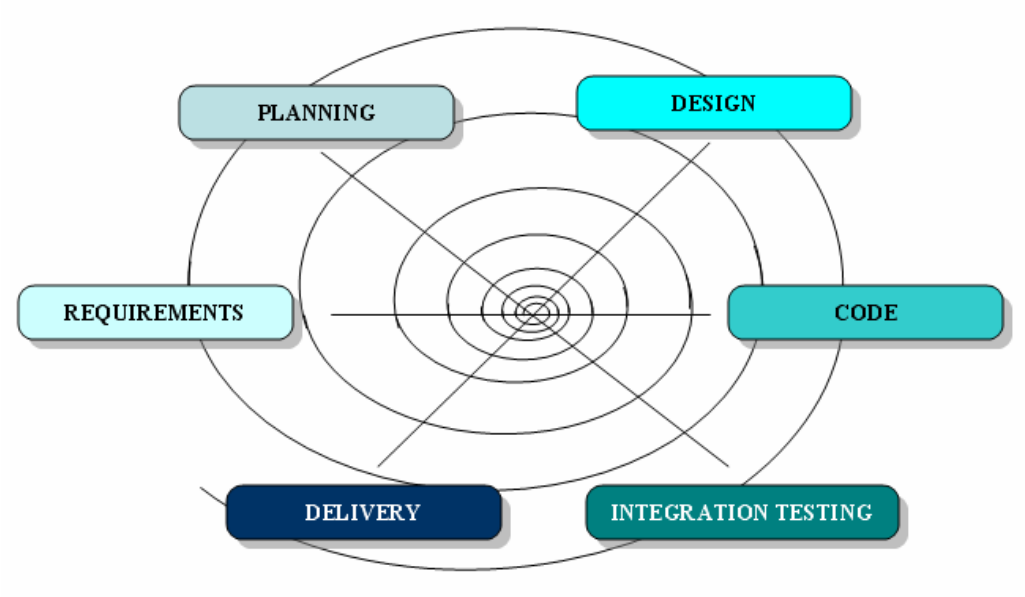

Figure 2.3 Spiral [PRE04]

2.1.6.2.1 Advantages of Spiral

The spiral is very realistic to the real world of building software applications [PRE04]. The model is understandable by most development teams and the customer. The model allows for reactions to risks that may be in development of the product.

2.1.6.2.2 Disadvantages of Spiral

The spiral model is often difficult to convince the customers that the evolution of the spiral is controllable [PRE04]. The model requires a lot of risk assessment experience and expertise. If a key risk remains undiscovered and unmanaged then there can be major issues [PRE04].

2.1.6.3 Incremental

The incremental process model uses the same steps as the waterfall process model but the steps are much smaller than the waterfall steps. Development of this model was very similarly to the spiral model recognizing that requirements are not complete at project start [PRE04]. Rather than going in a cyclical cycle like the spiral model, the incremental model has something similar to multiple waterfalls within the model as shown in figure 3.3. The incremental model also allows for starting with a very basic application and adding future functionality in subsequent updates [PRE04].

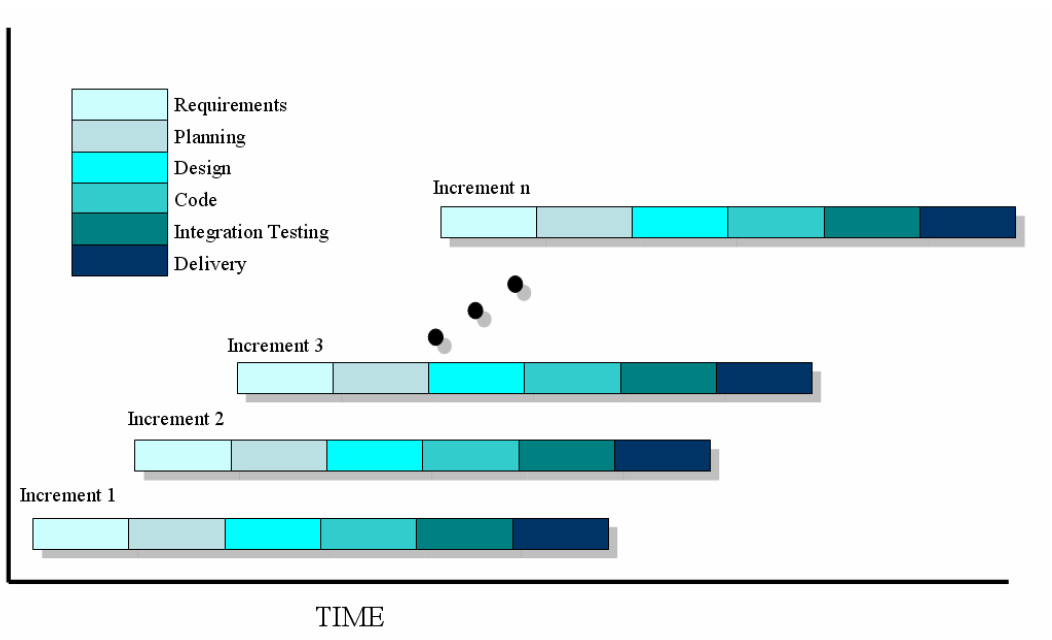

Figure 2.4 Incremental [PRE04]

2.1.6.3.1 Advantages of Incremental

The advantage of the incremental model is the beginning of a project normally has issues with staffing and the model allows for the beginning of the project to start with a smaller development team. If early increments are well received, then staffing can be added for future iterations [PRE04].

2.1.6.3.2 Disadvantages of Incremental

The incremental model assumes that there is a delivery with each iteration and the earlier iterations are stripped down versions, which may not be functional. The product may require specialized hardware for development and early increments may be earlier than the available hardware requiring special workarounds [PRE04].

2.1.6.4 Agile

Development of the agile process model was due to realizing that the customer was not always pleased with the product. Two different things were causing this: 1) the customer did not give all of the expected requirements and 2) engineers did not always understand the requirements [PRE04]. Extreme Programming (XP) is the most widely known and used agile process. This model has the customer involved throughout all of the steps and giving continuous feed back on the product. This model allows corrections and updates with each iteration of the product [PRE04]. This is an incremental process except each iteration is about one to two weeks in length. With XP programming the steps are only planning, design, coding and testing.

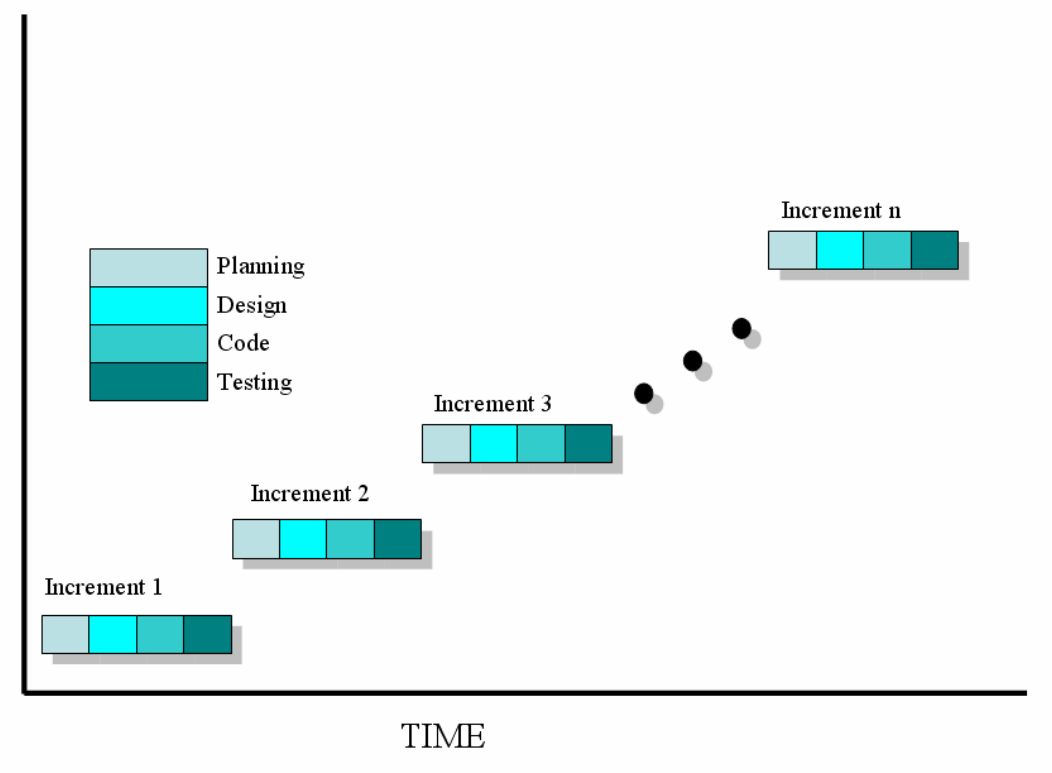

Figure 2.5 XP [LOF05]

2.1.6.4.1 Advantages of Agile

The model involves the customer earlier in the process, which helps in getting all of the requirements from the customer and fewer misunderstandings [LOF05]. XP uses what pair programming, which is where development requires two developers sitting side-by-side at a workstation. One developer is writing code and explaining what steps used; the other is observing, providing input and strategic advice. The two developers are to switch roles frequently while developing the code [LOF05]. The developers periodically split and regrouped so that the pairs are not always the same two developers. Paired programming reduces the number of mistakes and promotes the spread of knowledge across the development team [LOF05].

2.1.6.4.2 Disadvantages of Agile

The model involving the customer early can cause a constant flow of requirements, which can result in the product never completing [PRE04]. Management can view paired programming as a waste of resources.

2.1.7 Capability Maturity Model Integration^{SM} (CMMI^{SM})

Carnegie Mellon University (CMU) Software Engineering Institute (SEI) developed the Capability Maturity Model (CMM) in 1987 with assistance from Mitre Corporation by request of DoD [HU98, BIB02, LI02, GAO01]. The model evaluates and determines the level of maturity of a given organization and provides Software Process Improvement (SPI_2) practice [BIB02]. The model measures how well organizations were following their processes for development of their software applications. SEI worked to develop several maturity models in order to improve management in different areas [GAO01]. A few of the models were Software CMM® (SW-CMM®), Software Acquisition CMM® (SA-CMM®) and Systems Engineering CMM® (SE-CMM®) [GAO01]. Other organizations have since built many models for use with other processes besides software engineering.

2.1.7.1 SW-CMM

The SW-CMM was the first maturity model developed and was designed to improve software development processes [GAO01]. The maturity model was designed with five levels one being the lowest level and five being the top level. The higher level the organization was on the maturity model, the higher expectation of the project achieving their cost, schedule, and quality goals [GAO01]. There was an implication

that with maturity the organization's software processes were well defined, managed, controlled and effective [BIB02]. Each level of the model adds an additional component that is a key item for that level [BIB02].

2.1.7.2 SA-CMM

The SA-CMM was to improve the software acquisition process [COO02, GAO01]. Development of this model was from SEI's experience learned while developing the SW-CMM. SA-CMM follows the same five-level architecture [COO02, GAO01]. This model was to be a guide for acquiring products versus actually developing them. An individual acquisition starts with defining and planning of system need, which may start before establishment of a project office [COO02]. Therefore, the SA-CMM will include pre-contract award activities, which could include the solicitation package, initial set of requirements, and source selection. The acquisition ends when the contract for the products is completed and delivered [COO02]. Like the SW-CMM, SA-CMM must contain certain key processes and satisfy them to achieve the next maturity level. This model went through several iterations prior to rolling into the CMMI [COO02].

2.1.7.3 SE-CMM

Development of the SE-CMM was a means to assist organizations with their systems engineering by describing essential elements of organizations, which must exist in order to have a good systems engineering [BAT95]. Development of the model was using SEI's experience learned while developing the SW-CMM. This model also follows the same five-level architecture that was in the SW-CMM. This model does not

specify particular process or sequence, but it does provide a reference for comparing actual systems engineering practices against essential elements [BAT95]. The SE-CMM provides overall depiction of ideology and design, suggestions for suitable use, practices, and images of attributes of the model. SW-CMM was to provide success in a driven and contractually negotiated market area; an area determined by how competently an organization translated client needs into a product and how the product successfully met needs of the customer [BAT95].

2.1.7.4 CMMI

In 1997, CMM was changed to the CMMI by a team lead by DoD, SEI and industry to integrate the maturity models with associated products [LI02, GAO01, and CSE99]. The CMMI combines SEI and Electronic Industries Alliance models into a single model for use on enterprise-wide process improvement. This model was to be a benchmark for improving software development cost and schedule [LI02, GAO01, and CSE99]. SA-CMM features left out of the model were intentional, so software development processes could be concentrated on at the process start [GAO01]. The CMMI has five levels of maturity against which companies are measured, and organizations must sequentially step through each one in order [LI02]. The DoD requires that there be a formal validation of the processes with a site inspection for certification [LI02].

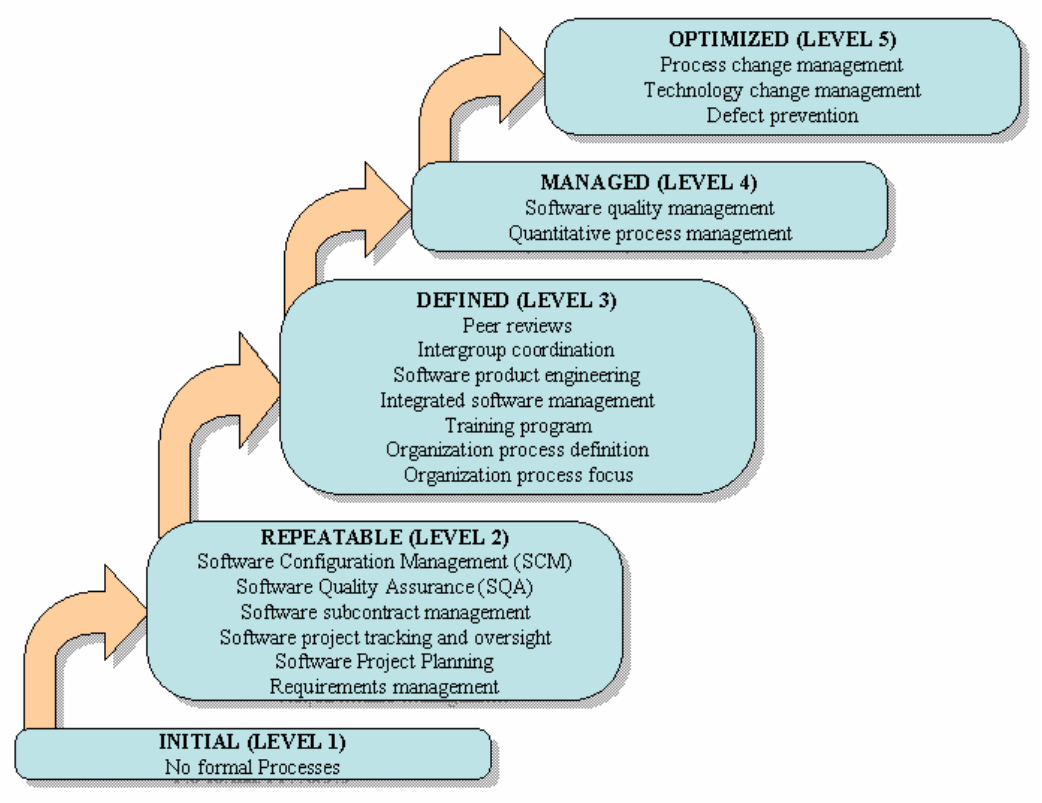

Figure 2.6 The Key Process Areas by Maturity Level [PAU93, CSE99]

2.1.7.4.1 Initial (Level 1)

The initial level is the beginning point of CMMI. When organizations are at this level, they are considered to be at the beginning of establishing basic development and management processes. Organizations are often struggling with basic management issues [CSE99]. Organizations normally do not have a stable environment for developing and maintaining software programs [BIB02, PAU93, and CSE99].

Processes, which are in place, are typically there due to a very influential manager or software team member [PAU93]. Organizations will often abandon any existing processes or procedures during a crisis since there may not be a clear reason to stay with these processes. The team will often resort to “code and testing” processes

[PAU93]. When this happens, success is entirely dependent on there being exceptionally seasoned software teams [PAU93].

The organizations processes are unpredictable because they do not hold to a single process and they may change several times during a project [BIB02, PAU93]. These changes have major impacts on cost and schedule. Performance on projects is purely dependant on the capabilities of the individuals on the team [PAU93].

Organizations at this level are in the process of Ad-hoc or chaos [BIB02, PAU93, BER03, GAO01, and CSE99]. They may use some Key Practice Areas (KPAs) required to be at level two and above but do not use all of them [LI02].

2.1.7.4.2 Repeatable (Level 2)

The Repeatable Level is where policies for managing software projects and procedures for implementing those policies are established [BIB02, PAU93, and CSE99]. New projects are planned using similar project experiences [PAU93]. Software project management processes are required at this level to be effective, which will allow for repeatability of successful practices from earlier projects on new projects [BIB02, PAU93, and CSE99].

The basic software project management controls are present for an organization to achieve level two effectively [BIB02, PAU93, CSE99]. Future project estimates have more accuracy using the visibility of prior similar projects. The projects cost, schedule and functionality are tracked and can be adjusted when problems arise within the projects [BIB02, PAU93, and CSE99]. Organization standards have documentation and faithfully followed [PAU93].

2.1.7.4.3 Defined (Level 3)

The Defined Level is where the standard process documentation exists for both the software engineering and management [BIB02, PAU93, and CSE99]. Process documentation is changed, as needed, to help organizations become more effective with each project. Processes are stable, repeatable, and implemented throughout the whole organization [BIB02]. Software engineering process group (SEPG) is responsible for software process activities [PAU93]. Standard process tailoring provides support to unique projects [PAU93]. Some of the major processes are the organization process focus, organization process definition, training programs, integrated software management, software product engineering, intergroup coordination, and peer reviews [BIB02].

2.1.7.4.4 Managed (Level 4)

The Managed Level is where the organization sets software products and processes goals on quantitative quality [BIB02, PAU93, and CSE99]. A software process database established for the whole organization to collect data and analyze data from the projects' well-defined software processes [BIB02, PAU93, and CSE99]. Organizations constantly take measurements to evaluate processes and predict quality trends within quantitative bounds [BIB02].

2.1.7.4.5 Optimizing (Level 5)

The Optimizing Level is where the whole organization focuses on continuous improvements of the processes [BIB02, PAU93, and CSE99]. Organizations can identify process weaknesses and strengthen them proactively to prevent product defects [BIB02, PAU93]. Data on software processes are key to performing cost benefit analyses for technology change management, reducing defects and software process change management [BIB02, PAU93, and CSE99].

2.1.7.5 Implementation cost of CMMI

The expense of implementing CMMI can vary from company to company. One company invested just over one million dollars in its process improvements [HER94]. Each level of the maturity model can take a considerable amount of time and training. One company now spends approximately 67 hours per person per year to achieve process improvements [HER94]. Some companies will stay longer at Level one but will move up quicker after reaching Level 2. The rapid increase is due to implementing many items for other levels while trying to reach the next level [LI02]. The DoD requires that a formal validation be performed by a site inspection for certification [LI02]. Figures 2.7 and 2.8 show how much cost companies have put into SPI_2 each year and how long they have been involve with SPI.

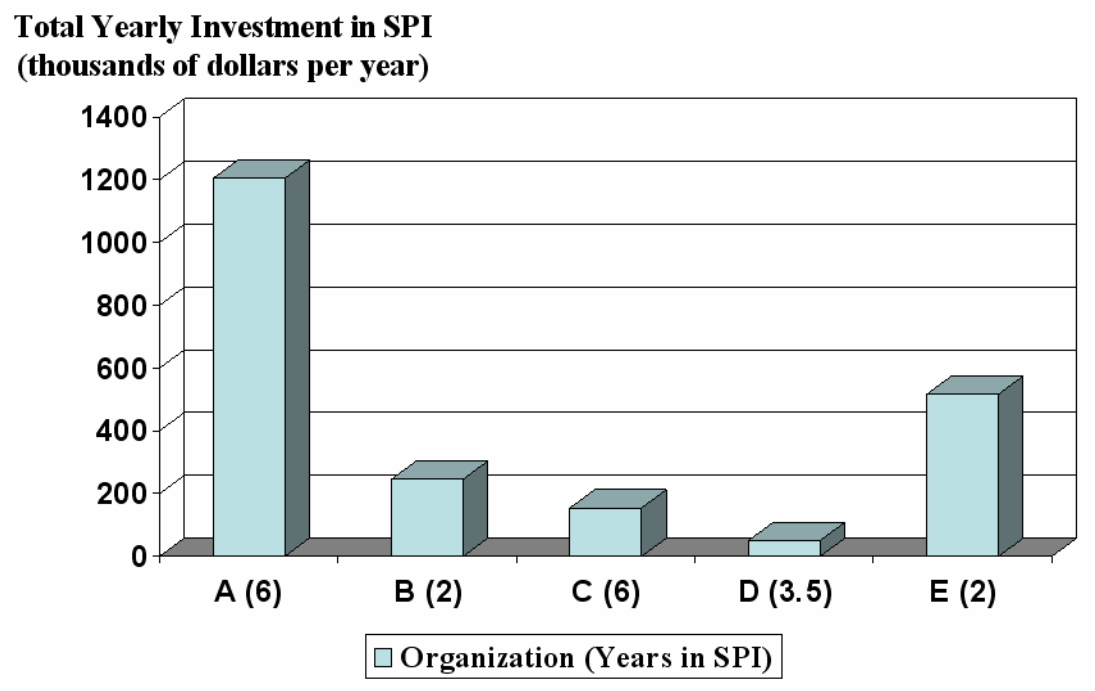

Figure 2.7 Thousands of Dollars per Year Spent on Software Process Improvement (SPI_2) [HER94]

Figures 2.7 shows organization A spent a total of 7,218,000 dollars (1,203,000 per year) over six years to implement software process improvements. Organization B spent 490,000 dollars (245,000 per year) over two years. Organization C spent 930,000 dollars (155,000 per year) over six years. Organization D spent 171,500 dollars (49,000 per year) over three and one half years. Organization E spent 1,032,000 dollars (516,000 per year) over two years [HER94]. The organizations agree that the cost was more than they anticipated [HAR00, HER94].

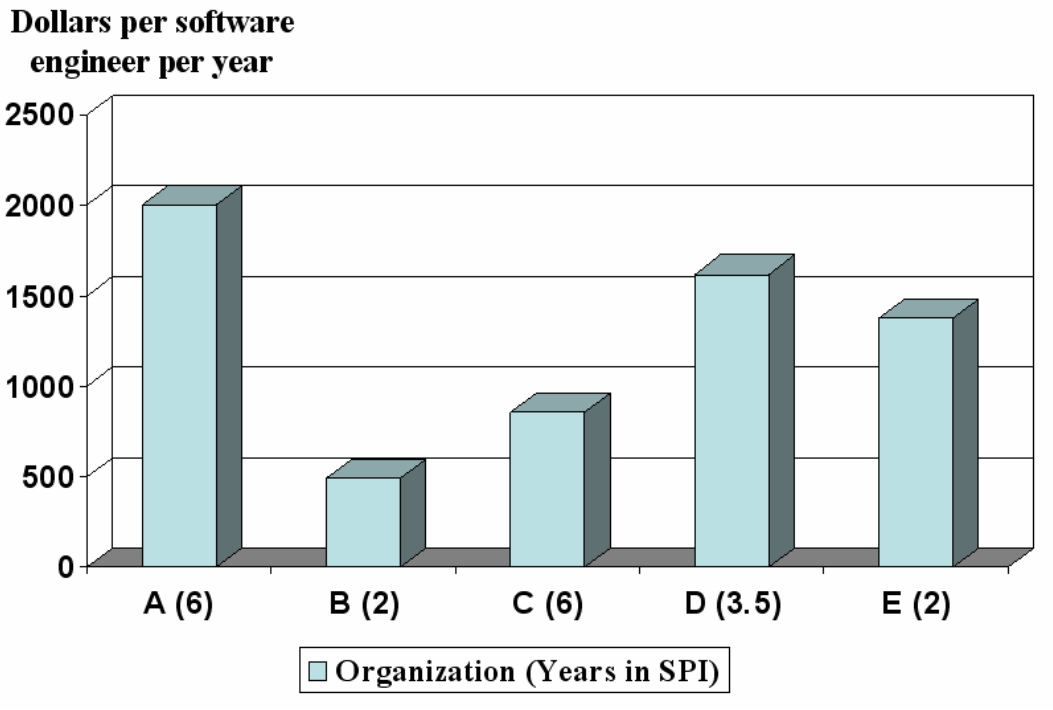

Figure 2.8 Dollars per Software Engineer per Year Spent on Software Process Improvement (SPI_2) [HER94]

Figures 2.7 shows organization A spent 2,004 dollars per year per software engineer; organization B spent 490 dollars per year per software engineer; organization C spent 858 dollars per year per software engineer; organization D spent 1,619 dollars per year per software engineer; and organization E spent 1,375 dollars per year per software engineer. The largest organization and the smallest organization are the two organizations that spent the most per software engineer [HER94].

CHAPTER 3

EXPERIMENT DESIGN

3.1 Problem Statement

Software development cost has grown exponentially over the last forty years [GAO01, HAR00]. There are many projects never finished due to extreme budget overruns or delivery slippages. Many companies are finding that their software alone, in-house built or outsourced, is usually more than 50% of the total budget of IT. [LI02]. DoD has been having difficulty in effectively managing information technology which is critical to its ability to accomplish its mission [GAO01]. Cost overruns being caused by software slippages at an average of $225,000 reported by Peat Marwick Mitchell and Co in 1987 [HU98].

3.2 Hypothesis

Implementing the CMMI will provide increased productivity to companies. Each level in the model that a company achieves will provide an increase in productivity. The increased productivity will allow the project to complete earlier than they were prior to implementation; bringing in extra revenue to companies since fewer projects receives cancellations due to cost overruns.

3.3 Investigation

The investigation will include researching surveys conducted by many organizations. All of the data will be collected and compiled to analyze what the results

show that the companies have experienced over the years with the implementation of CMMI models.

CHAPTER 4

EXPERIMENT

4.1 The Application of Modern Software Engineering Practices to Control Engineering

Brian Berenbach and Peter Spool, in "The Application of Modern Software Engineering Practices to Control Engineering" [BER03] indicate in 1980, hardware was 60 percent of the control system cost and engineering was only 30 percent. Today, cost has change to where 35 percent is hardware cost and 50 percent is engineering. Manufacturing organizations have known that manufacturing process used is largely responsible for cost and product quality. Organizations are beginning to realize this is the same case with producing software.

The mean gain found when an organization started applying CMM has been about 35 percent per year for the first few years. The mean reduction of time to market has been about 19 percent and reduction in defects has been 39 percent per year with one organization reporting a 94 percent reduction in defects. An often overlook task is mapping of overall processes to aid in tracking and oversight of the project. Tracking and oversight is to provide a visibility into actual progress of the project so that management can take effective actions when needed. Actions may include revising the software development plan, revising remaining work, or taking other actions to improve performance.

A common misconception of Software Quality Assurance (SQA) is that SQA is only for testing the product when SQA is usable for much more and should provide verification and validation across the project life cycle. CMMI level three is the level where peer review is located but peer review is empirical to have significantly higher quality of software and a pronounced reduction in product defects.

4.2 Software Configuration Management

Edward Bersoff, Vilas Henderson, and Stan Siegel, in "Software Configuration Management" [BER78] indicate discipline is a key item needed within software development. The problem has been that software has been behaving in an undisciplined manner. With SCM, the software product can achieve integrity, which the customer desires. SCM implementation not done similar to traditional Configuration Management (CM) for hardware has been the major issue. One issue that management has is "How do you effectively manage something that you can not see?" SCM has failed when it does not follow the direct analogy from hardware CM practices.

There are two basic forms of software non-executable form and executable form. The non-executable form exists early in the process where the executable form does not. The non-executable form may exist in many different languages while being developed even pseudo-code. The executable form will exist only in a language, which is compliable. The life of software is similar to a biological system and goes through many stages

The major difference between hardware and software is building software as a one-of-a-kind and mass production of software is merely copying program code. Finding and correcting flaws is another area where hardware and software differ.

Primary role of software configuration control is to provide administrative mechanism for controlling (approving/disapproving) all change proposals throughout system life cycle. Software configuration control is change proposal processing.

4.3 A Survey of Industrial Experiences with CMM and the Teaching of CMM Practices

Erol Biberoglu and Hisham Haddad, in "A Survey of Industrial Experiences with CMM and the Teaching of CMM Practices" [BIB02] indicate that in 1987, the DoD requested the development of a five-level CMM model. CMM is to be a framework describing the key elements for effective software processes and is a guide for improving software development practices through planning, engineering, management, and maintenance. Process maturity means that organization's software processes are well defined, managed, controlled, and effective. Each level of the model focuses on one significant module of the process.

In 1996, James Herbsleb and Dennis Goldenson of SEI conducted a survey of organization using CMM to improve their software processes. The survey showed a definite relationship between maturity and performance. Organizations with a higher maturity level showed more improved performance than lower organizations.

An earlier study conducted by SEI of 13 organizations reviled that organizations achieved better performance cycle time, defect density and productivity with implementation of CMM. The benefit-to-cost ratio ranged from 4:1 and 9:1 indicating

CMM based process improvement pays off in the end. One characteristic reviled was CMM helped an organization identify process weaknesses and technical areas where there needs to be improvements.

The implementation of CMM can identify process weaknesses and technical areas where organizations need to improve. However, the implementation of CMM without applying the Business Process Reengineering (BPR) can be counterproductive. Implementation of CMM in small organizations has shown not to be very effective and often more expensive than it is worth. European organizations have relied on ISO9001 for process improvements.

4.4 DOD INFORMATION TECHNOLOGY: Software and Systems Process Improvement Programs Vary in Use of Best Practices

United States General Accounting Office, in "DOD INFORMATION TECHNOLOGY: Software and Systems Process Improvement Programs Vary in Use of Best Practices" [GAO01] Report to the Chairman and Ranking Member, Subcommittee on Readiness and Management Support, Committee on Armed Services, U.S. Senate indicates that the government recognized the growth of information technology budget to about 20 billion dollars and tens of billions of dollars more budget for technology embedded in weaponry. DoD relies heavily on these new software-intensive weapons and need to control quality, cost, and schedule. DoD recognized the importance of controlling processes in order to effectively control quality, cost, and schedule of projects. Public and private organizations, which have been successful, have adopted and implemented software/systems process improvement programs. US Aviation and Missile Command (AMCOM) Software Engineering Directorate (SED)

increased productivity ratio from 1.30 to 2.48 just by moving from CMM level two to level three on new development products.

4.5 Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development

Donald Harter, Mayuram Krishnan, and Sandra Slaughter, in "Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development" [HAR00] indicate that IT firms do not consistently follow their practices due to the dynamic nature of commercial software development environment. The customers will frequently change requirements after production design has started and still expect delivery of the product on time. Firms have adopted time-based competition strategies to reduce cost of the product with reduced product development time without sacrificing quality. Important issues with process improvements are achieving higher quality, reduced cycle time, and lower cost. Typically, higher quality means longer cycle time and software managers would prefer to sacrifice quality than to miss schedule deadline. Manufacturing view is that quality, cost, and cycle time are complementary so improvements in quality will result in improved cycle time and productivity.

The study performed on a major IT firm on 30 software projects over a 12-year period showed that improvements in process maturity lead to higher quality resulted in increased effort and reduced cycle time. The study reviled for each one percent improvement in process maturity resulted in a 0.32 percent net reduction in cycle time and a 0.17 percent net reduction in development effort. This result was taking into account any positive and negative effects directly or indirectly through quality. The

study showed that increasing from CMM level one to level two had a cycle time reduction of 183-calendar-days and 23-person-months reduction in development time. The study reviled moving from level two to level three reduced cycle time an additional 90-calendar-days and a 12-person-months reduction in development effort.

Prior literature showed that the focus for development of models was to analyze defects, to prevent defects and to predict reliability of software products. Nandakumar and others argued about whether poor quality in manufacturing systems would result in higher defect rates and increase production development time. Research showed that in the absence of interrelationship understanding between quality, time to market and cycle time would result in elimination of needed inspections with the belief of saving time resulting in increased cycle time, increased time to market, and decreased quality.

Swason, Abdel-Hamid, and Madnick performed several studies, which indicated when defects were found in early stages the defect required less time to fix than if the defect was found in a later stage. This study supported the thought that quality, cost and cycle time are complementary.

Some studies showed increased maturity levels resulted in higher quality and increased cycle time. The improvements to quality outweighed the marginal increases in the cycle time.

4.6 Benefits of CMM-Based Software Process Improvement: Initial Results

James Herbsleb, Anita Carleton, James Rozum, Jane Siegel, and David Zubrow, in "Benefits of CMM-Based Software Process Improvement: Initial Results" [HER94] indicate teams initially skipped inspections to improve time but found that defects

increased and was talking four times longer in time to repair the defects. The teams determined that for each stage a defect remains in the product, cost to remove it doubles. Improvements to the cycle times have been difficult to quantify because of changes in headcount but where it was quantified, cycle times were cut in half. The quality has decreased defects by 7 to 10 percent per year because of the SPI_2 program.

Inspections alone are not a certain remedy for overcoming serious software engineering process flaws. Inspections before unit test are best since code is smaller and easier to inspect. Omitting inspections before unit testing can result in code being costly to detect and repair defects.

Tinker OC-ALC estimates it has saved over four million dollars as a direct result of process improvements. The primary savings is from cost avoidance such as rework. Savings is a return of more than four dollars for every dollar invested in software process improvements. Process improvements scientifically improved employee satisfaction and reduced employee turnover.

4.7 Software cost estimation using economic production models

Qing Hu, Robert Plant, and David Hertz, in "Software cost estimation using economic production models" [HU98] indicate there are problems of software cost overruns and schedule slippages. A 1984 study reviled 72 software projects in 23 major US corporations had a median cost overrun of about 34 percent, an average of 67 percent and average schedule slippage of about 22 percent. The average cost overrun was about 225,000 dollars and schedule slippage of about three-calendar months. When comparing several major software cost models, results showed the average magnitude

of relative errors (MRE) ranged from 85 to 772 percent with many in the 500 to 600 percent range. The cost models were using data from the 1960s and 1970s, which the data could not determined if it would be accurate in today's complex software environment.

4.8 Change management needs integrated process and configuration management

Gregor Joeris, in "Change management needs integrated process and configuration management" [JOE97] indicates that change management is a core problem of software development. Change management to software documents is managing the change process and managing all evolving software system artifacts. Like every engineering process, software process has complexity of both product and process. Processes are in a dynamically changing environment. A key item of a well-defined software development and maintenance process is SCM.

4.9 Software process management of top companies in Taiwan: a comparative study

Eldon Li, Houn-Gee Chen, and Tien-Sheng Lee, in "Software process management of top companies in Taiwan: a comparative study" [LI02] indicates how well one manages software engineering processes is key to developing a quality software product. The selection of the top 1000 companies in Taiwan for a surveyed consisted of 667 manufacturing, and 333 service companies. The initial and second mailing yielded a 13.8 percent response rate. The survey results showed that organizations did not always perform only the activities identified as level one but would perform activities identified at all levels simultaneously. Because of this, the

perception is organizations will remain at level one longer than expected. The Taiwan's business industries are still in its infancy stage with software process management. Most organizations, 78.3 percent, did not implement more than 50 percent of the key practices and the average was 35.4 percent achievement. This may be due to the lack of software engineering training among software professionals in these organizations. The Taiwan's organizations were poor about gathering code and test error data, controlling software complexity, regression tests, and human resource usage. The organizations did not appear to be interested in these and seemed more interested in value-added activities such as analysis, design, programming, and installation. The Taiwan organizations do not emphasize quality management techniques and processes. Few organizations measured software complexity (8.5%) and software size (15.4%) which reinforces the opinion that proper documentation of project effort was not performed and could not be quantitatively estimated in most organizations. Management will need to change the philosophy on quality and eliminate the thought that "quality is at the expense of productivity." It needs to recognize that without quality, productivity means nothing, and to carry out key CMM practices adequate resources must exist in order to manage effectively software process, and produce quality software. The Taiwan organizations did not have the ability to make error prediction and to prevent errors from happening. Organizations were not able to design training programs for the staff to learn how to reduce human errors in design, code, and test activities. Front-line managers were not empowered to effectively manage groups and eliminate communication gaps.

4.10 Capability maturity model, version 1.1

Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, in "Capability maturity model, version 1.1" [PAU93] indicate in mature organizations, managers monitor software product quality and customer satisfaction. The mature organization's software process provides a basis for determining the ability of a project to meet its goals. Level one organization will often miss their original schedule delivery where a level five organization should be able to meet its schedule delivery.

4.11 An object-oriented model of software configuration management

Hal Render and Roy Campbell, in "An object-oriented model of software configuration management" [REN91] indicate that SCM spans the entire life cycle of software, from initial design through release and maintenance. SCM is concerned with identifying, organizing, and controlling changes to the components of a software project. The project derived object identification and to automate their source is one aspect of a SCM system.

CHAPTER 5

EXPERIMENT ANALYSIS AND RESULTS

The collected surveys showed substantial improvements within many software companies. Survey revealed 16.9 percent of the companies found that empowering their front-line supervisors helped in this process but the rest of the companies found this hard to implement [LI02]. The empowering of the front-line supervisors removed many communication gaps. Ninety percent of the companies reported improvements and 4 percent reported a negative affect [BIB02]. The surveys revealed 4 percent of the companies felt that the CMMI implementation resulted in a loss of productivity [BIB02]. The companies (136 out of 138) found improvements by implementing activities and processes from several levels of the CMMI model. However, this prevented them from advancing to the next level of the model until all of the activities of the next level were implemented and only 2 out of 138 achieved higher than level 1 [LI02]. In organizations which have not understand the relationship between inspections and the time to remove defects, management has removed some inspections in order to save time and found that the removal of the inspections had adverse effects on the schedule [HAR00, HER94]. A key item to improving the process was commitment from management [HER94].

5.1 Increased Productivity

Studies have revealed that implementing CMMI has increased productivity in about 90 percent of the companies. One survey showed between a 4:1 and 9:1 improvement in productivity [BIB02] while another showed a median gain of 35 percent [BER03]. Survey reveals that some believe that implementing CMMI without applying Business Process Reengineering (BPR) would be counter productive. The surveys reveal that 90 percent of the companies disagree that CMMI has an adverse effect on productivity [BIB02]. One survey found that a 1 percent implementation of process maturity model results in a 0.32 percent reduction in cycle times [HAR00]. According to the US Aviation and Missile Command (AMCOM) Software Engineering Directorate (SED), they have increased their productivity ratio from 1.30 to 2.48 just by moving from level 2 to level 3 [GAO01].

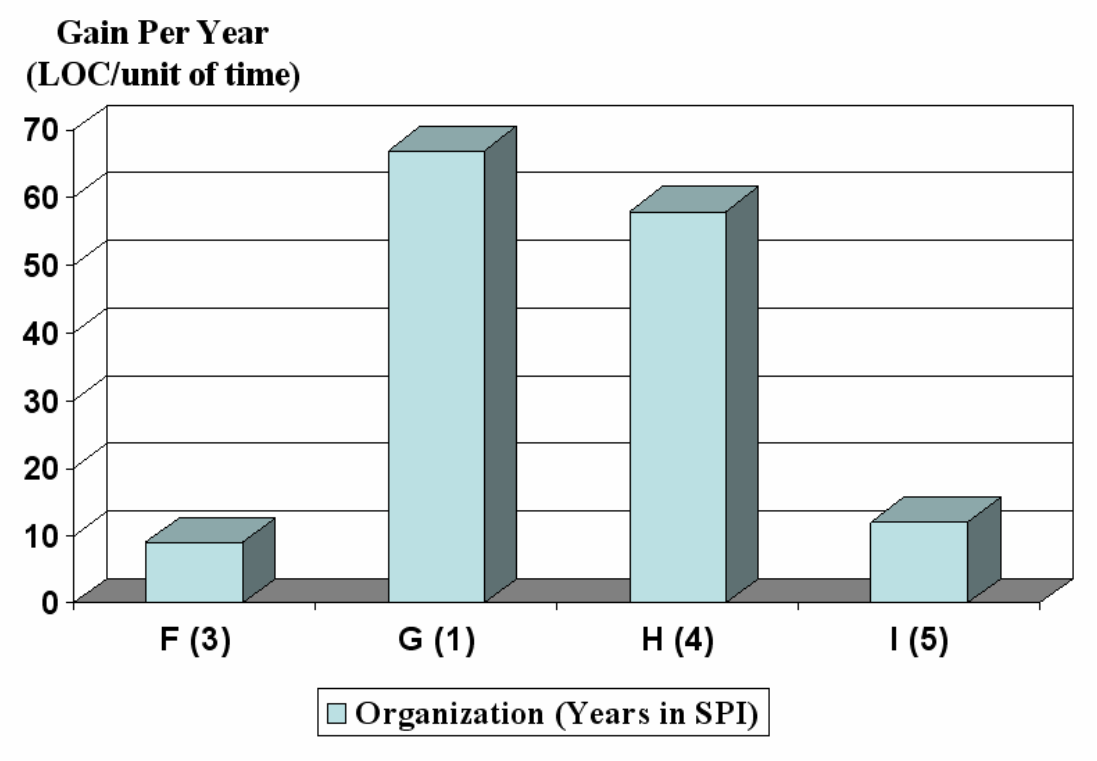

Figure 5.1 Percentage Gain per Year in Productivity [HER94]

Figure 5.1 shows how the organizations have benefited from performing software process improvement and advancing to higher levels of CMMI. Organization F has a 9 percent gain per year for LOC (lines of code) per unit of time; organization G has a 67 percent gain; organization H has a 58 percent gain; and organization I has a 12 percent gain [HER94].

5.2 Reduced Production Times

Some survey results have not shown an actual reduction in production time schedule but rather the organizations are now meeting the planned production time schedules. Previously, the completion schedules were generally sliding to later dates [HER94]. Surveys have shown an average of 22 percent schedule slippage prior to

implementing CMMI [HU98]. The estimates of the effort are now more realistic to what an organization can realistically accomplish. Some reported as much as a 50 percent reduction in production times. Over the last two years, the actual cost has been running less than the budget and has increased the EVMS CPI measure substantially [HER94].

5.3 Reduced Defects

Studies show that the quality of the delivered product improved greatly by implementing the CMMI. Defects reduce as an organization moves up in the maturity model. This is a result of both improved software code and defects found early in the process cycle. One of the organizations decided to skip quality inspections during some of the process steps to save production time and found that this increased defects going into the next process step. When the defects found in any given process step the cost is 2 to 4 times higher than finding the defect in the previous process step [HAR00, HER94]. So if a defect is found in code the defect is as much as 4 times more costly than if the defect were found in design and if the defect is not found until integration testing then the defect is as much as 16 time more costly. Surveys have shown that by implementing the process models the defects decreased from 0.21 per KLOC to 0.14 or approximately 34 to 39 percent per year [BER03, HER94]. One company reported a 94 percent reduction in defects after implementing the CMMI [BER03]. The customers were reporting 7 to 10 percent fewer defects per year [HER94]. Implementation of Software Configuration Management (SCM) helped with introducing well-defined

change processes, which allowed better control over the configuration [BER78, JOE97, and REN91].

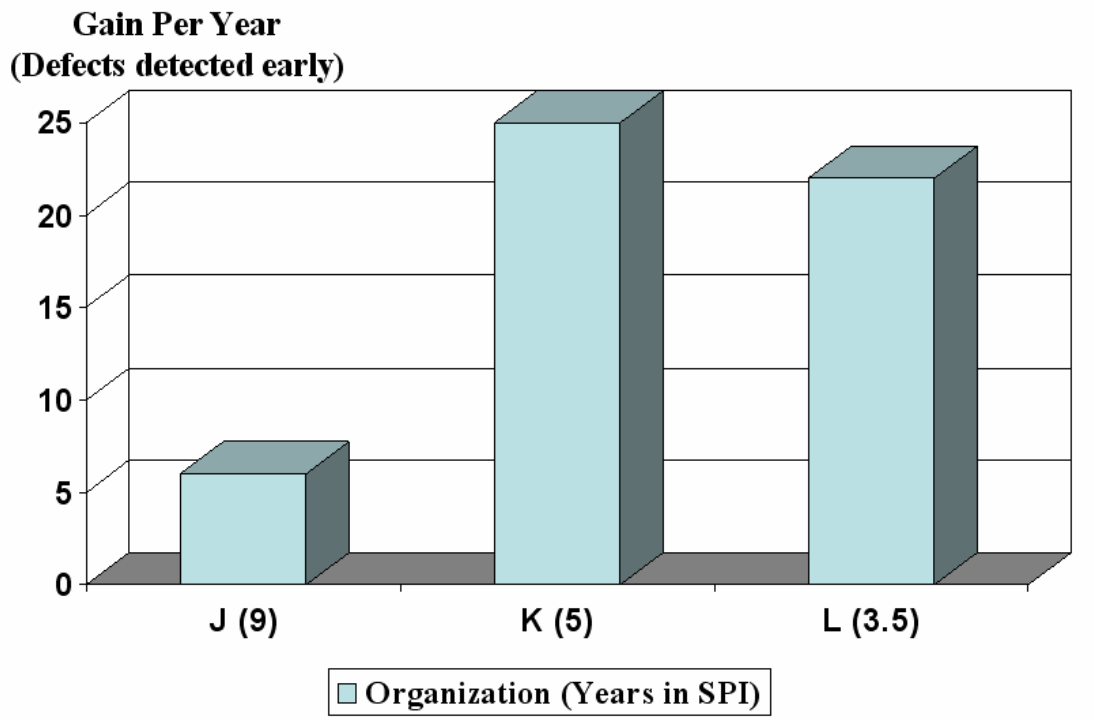

Figure 5.2 Percentage Gain per Year in Early Detection of Defects [HER94]

Figure 5.2 shows how the organizations have benefited from detecting defects early in the development process. The early defect detection represents an enormous savings with preventing rework costs. Organization J has a 6 percent per year gain by detecting the defects early; organization K has a 25 percent gain per year; and organization L has a 22 percent gain per year with implementation of software process improvements [HER94].

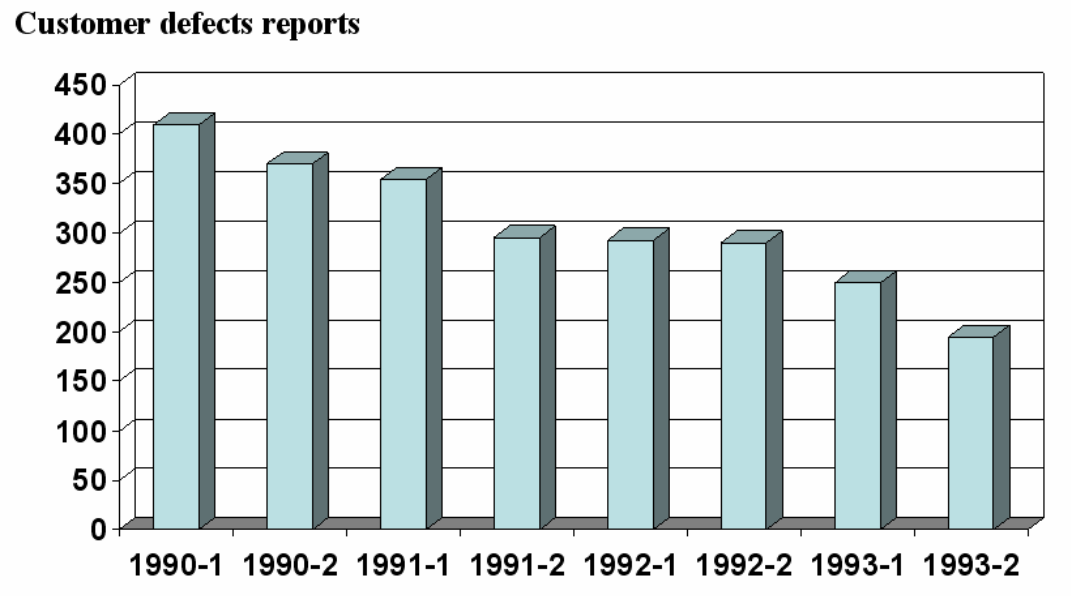

Figure 5.3 Mean Monthly Number of Customer-Reported Defects for Six-Month Periods [HER94]

Figure 5.3 shows the number of customer found defects reported. Since the implementation of inspections on the products, the organizations have reported a 50 percent reduction in fixes themselves and only 13 percent of inspects have found any software defects which makes this a larger benefit than originally thought. The attributed extra benefit is one of two things; the maintainers either are working more carefully knowing that their work will be inspected or to lower defect injection cause by other facets of the software improvement process [HER94].

5.4 Improved Estimates

The improved processes have provided organizations the ability to be more accurate with their cost and schedule estimates. Figure 5.1 shows that before the use of CMMI that the organizations were generally missing their targets and afterwards not

only did they estimate closer to reality but they were actually able to reduce their price [HER94].

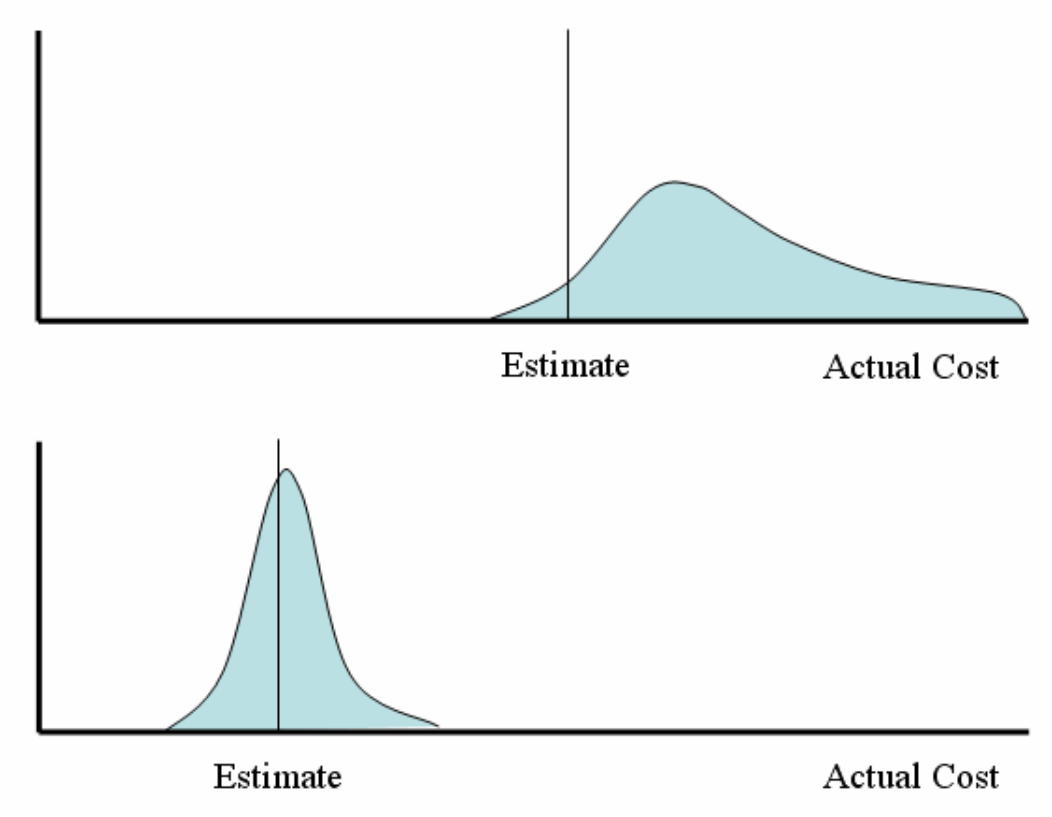

Figure 5.4 Theoretical Cost Estimating Outcome [HER94]

Figure 5.4 shows how the organizations original estimates are comparing to the actual outcomes of the projects with implementation of software process improvements. The organizations not only are meeting their estimates but they have reduced the times in their estimates [HER94].

5.5 Improved Cost Performance

The EVMS Cost Performance Index (CPI) shows an improved cost performance on the projects after the CMMI implementation [HER94].

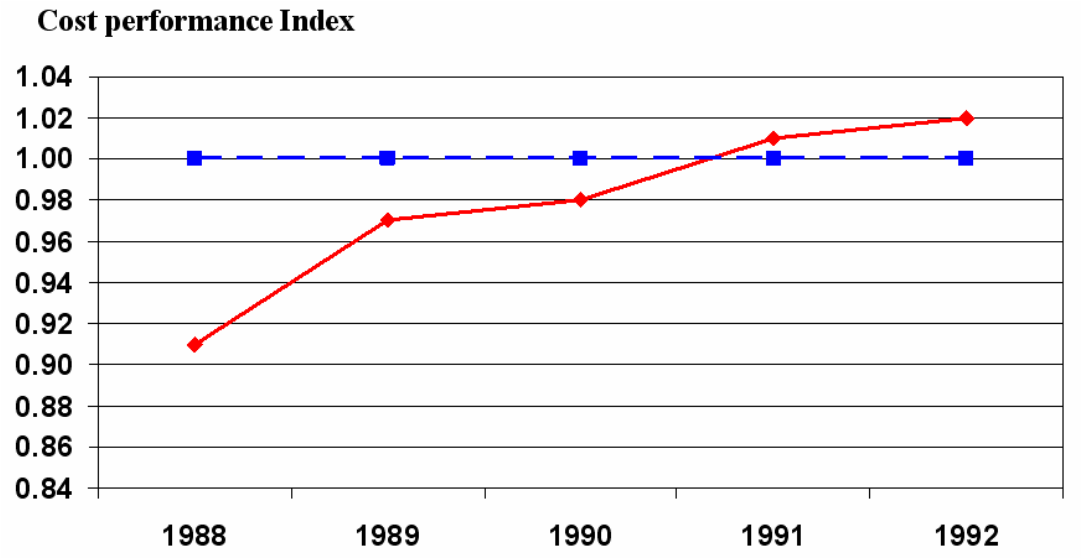

Figure 5.5 Cost Performance Index (CPI) [HER94]

Figure 5.5 shows how the organizations cost metrics have improved with implementation of software process improvement. The cost performance index is showing that the organization projects have gone from producing 0.91-dollar value per dollar spent to a 1.02-dollar value per dollar spent over the four-year period. This indicates that the customer is getting much more value for their money, which is an 11-cent increase per dollar, spent [HER94].

5.6 Improved Schedule Deliveries

The survey shows the deliveries of the products improved by 19 percent [BER03].

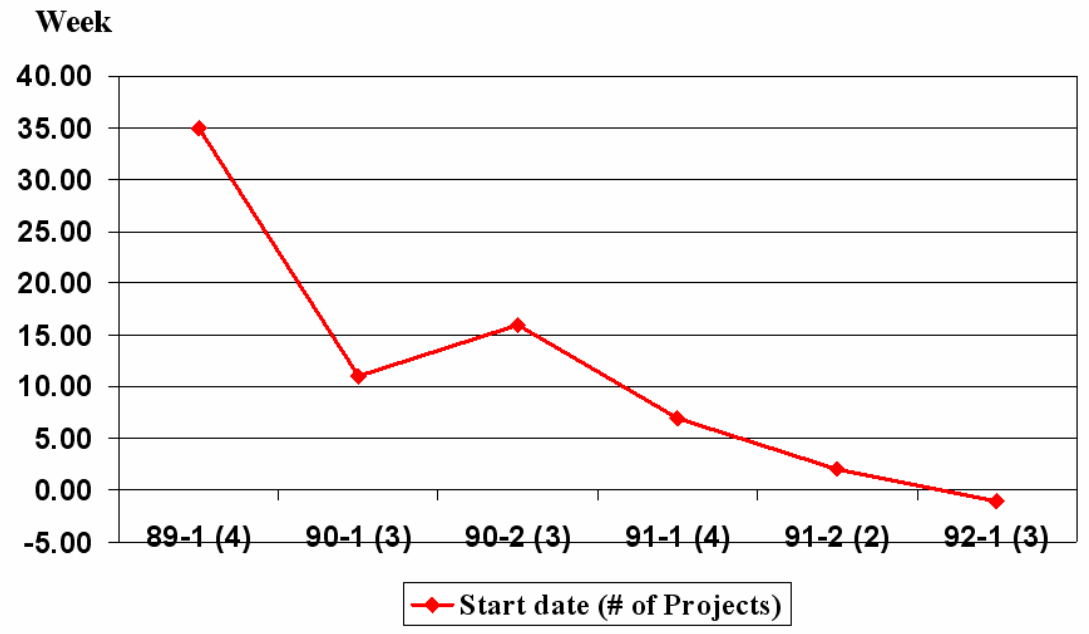

Figure 5.6 Average Difference Between Estimated and Actual Completion Times [HER94]

Figure 5.6 shows the difference between the actual completion dates and the original estimate. The projects were averaging a completion of 35 weeks behind schedule and are now completing 1 week ahead of schedule even with a shortened project duration time with implementation of software process improvement [HER94].

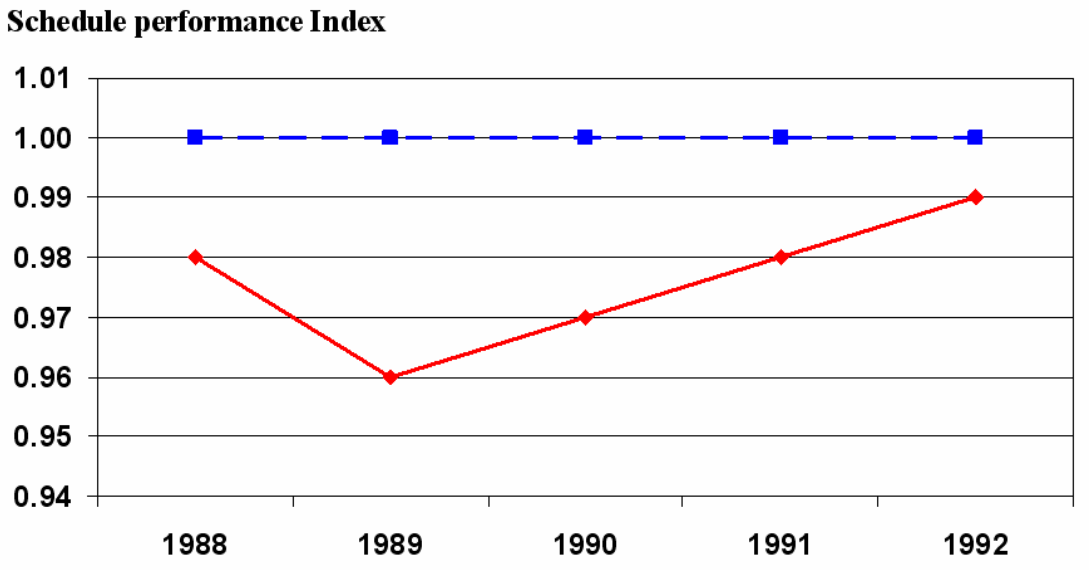

Figure 5.7 Schedule Performance Index (SPI_1) [HER94]

Figure 5.7 shows how the projects cost metrics on the schedule performance index have improved with software process improvement. The projects have a minor drop in schedule performance the first year after implementation but quickly returns after the processes become a little more mature. The projects have improved by 0.01 each year after implementation [HER94].

5.7 Increased Customer Satisfaction

Customers have been reporting fewer defects in products, the organizations are working closer to the customers, and the organizations believed that there is an improvement in customer satisfaction [HER94].

5.8 Savings

Implementing the CMMI is very expensive but the savings outweighs the cost in the end. Surveys have shown extensive savings. A summary of the reported savings from the companies surveyed are in the Table 5.1.

Table 5.1 Summary of Reported Savings [BER03]

Category	Range	Median
Total yearly cost of SPI activities	$49,000 – $1,202,000	$245,000
Years engaged in SPI	1 – 9	3.5
Cost of SPI per software engineer	$490 – $2,004	$1,375
Productivity gain per year	9% – 67%	35%
Early detection gain per year (defects discovered pre-test)	6% – 25%	22%
Yearly reduction in time to market	15% – 23%	19%
Yearly reduction in post-release defect reports	10% – 94%	39%
Business value of investment in SPI (value returned on each dollar invested)	4.0 – 8.8	5.0

5.9 Satisfied Employees

The survey revealed that the process improvements created better morale with improved understanding of the corporate mission and vision. There was evidence that there were fewer crises, less stress, better quality of work life, less employment turnover, less overtime work required and better organization communication reported because of software process improvement. The employees shared a sense of pride in their work [HER94]. The process improvements met with resistance in large corporations in many cases since the workers determined the improvements to be a means of downsizing. These benefits are frequently not considered since they are intangible and do not have an associated cost-benefit but needs to be acknowledged since they are important. There are two reasons these intangible benefits are important. First, when given new process alternatives with similar tangible returns then consider the intangible return. Second, these benefits are indicators of organizational climate or culture, which is a significant part of implementing continuous process improvements.

The one item that has not been widely addressed is how to adequately measure and place values on employee satisfaction changes [HER94].

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

Software process improvements are very expensive to implement in any organization and has cost many organizations more than expected in resources, time and money. The effort to reach each CMMI level requires a large amount of resources and time prior to the on-site inspection to certify the organization.

Cost benefits for implementing CMMI are substantial and worthwhile. An Organization ROI after implementing CMMI is approximately 35 percent per year with an improved schedule of 19 percent for a total ROI of about five times the investment [BER03]. Some organizations, measuring from level one, are reporting productivity increases of 300 to 500 percent by the time they reach level three and an additional 350 percent when they reach level five [KRA97]. Process improvements create an approximate savings of 4.7 million dollars per project [HER94].

Implementing CMMI provides any organization with benefits such as making them more competitive in the market place. Four percent of the organizations state issues with implementing CMMI. This could have been due to the size of the organization since there is a minimum cost associated with implementation of software process improvements. Small organizations have to evaluate the cost of implementing CMMI, the time recovering the investment, and determining if they can implement

while remaining competitive. Some organizations cannot remain competitive due to the large cost associated with implementation of CMMI [BIB02]. Small organizations can adopt some of the software process improvements helping the organization to have some benefits where it would be cost prohibitive to implement fully the CMMI.

Implementing CMMI reduces the number of defects in the product by 39 percent. The discovery of defects in the development process is earlier than before. Defects take six to 1000 times longer to fix if not detected until a later process step. Caught early, 10 percent fewer defects reach the customer. The customer is more satisfied and has more confidence in the product quality since there are fewer defects.

The CMMI improvements increase the employee moral by preventing crises. The employees are enduring less stress, working less overtime, improving communication, and taking on pride in their work; resulting in a lower employee turnover rate. These results are taking place even with some employees resisting for fear of downsizing in the organizations.

The DoD requires contractors of major acquisition projects with a procurement cost of over 2 million dollars to be certified as level 3 CMMI or higher. Therefore, any organizations desiring to work with DoD will need to implement CMMI.

Although CMMI is expensive and time consuming, an organization would greatly profit from the implementation of these software improvement processes. Organizations incorporating CMMI are more competitive, with fewer defects, with increased productivity, with reduced time to market, more satisfied customers and increased employee moral.

6.2 Future Research

Future research in software process improvements should perform new surveys to each of these companies and see what advancements have happened since the last survey was completed. In the surveys described in chapter five, none of the companies were above level three but many of them have probably reached level four or five. It is very likely that these companies now have much greater visibility of how software process improvements are working for their organizations.

REFERENCES

[BAT95] Roger Bate, Dorothy Kuhn, Curt Wells, James Armitage, Gloria Clark, Kerinia Cusick, Suzanne Garcia, Mark Hanna, Robert Jones, Peter Malpass, Ilene Minnich, Hal Pierson, Tim Powell and Al Reichner, "A Systems Engineering Capability Maturity Model, Version 1.1", CMU/SEI-95-MM-003, Software Engineering Institution, Pittsburgh, PA, November 1995.

[BER03] Brian Berenbach and Peter Spool, "The Application of Modern Software Engineering Practices to Control Engineering", Journal of Advanced Manufacturing Systems, Vol 2, No. 1, 2003, pgs 127-141.

[BER78] Edward Bersoff, Vilas Henderson and Stan Siegel, "Software Configuration Management", Proceedings of the software quality assurance workshop on Functional and performance issues, 1978, pgs 9-17.

[BIB02] Erol Biberoglu and Hisham Haddad, "A Survey of Industrial Experiences with CMM and the Teaching of CMM Practices", Journal of Computing Sciences in Colleges, Vol. 18, No. 2 December 2002.

[BOE95] Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0", International Society of Parametric Analysts, May 1995.

[CHA94] George J. Chambers, "Variance Analysis within C/SCSC Programs", Acquisition Review Quarterly, No 67, Winter 1994.

[COO02] Jack Cooper and Matthew Fisher, “Software Acquisition Capability Maturity Model® (SA-CMM®), Version 1.03”, CMU/SEI-2002-TR-10, Software Engineering Institution, Pittsburgh, PA, March 2002.

[CRA06] Lynn Crawford, Julien Pollack, David England, “Uncovering the trends in project management: Journal emphases over the last 10 years”, International Journal of Project Management, Vol 24, No 2, February 2006.

[CSE99] Center for Software Engineering, “Technical Briefing the Capability Maturity Model® (CMM®), October 1999.

[DOD96] “United States Department of Defense News Release”, Reference Number: No. 678-96, December 18, 1996.

[GAO01] United States General Accounting Office (GAO), “DoD Information Technology: Software and Systems Process Improvement Programs Vary in Use of Best Practices: GAO-01-116”, GAO Reports, 30 March 2001.

[HAR00] Donald Harter, Mayuram Krishnan and Sandra Slaughter, “Effects of Process Maturity on Quality, Cycle Time and Effort in Software Process Development”, Management Science, Vol 46, No. 4, April 2000.

[HER94] James Herbsleb, Anita Carleton, James Rozum, Jane Slegel, and David Zubrow, “Benefits of CMM-Based Software Process Improvement: Initial Results, CMU/SEI-94-TR-013, Software Engineering Institution, Pittsburgh, PA, August 1994.

[HU98] Qing Hu, Robert Plant and David Hertz, “Software Cost Estimation Using Economic Production Models”, Journal of Management Information System, Vol 15, No 1, Summer98, pgs 143-163.

[HYS99] Debbie Hysell, “ISO 9001: Traditions Before and After”, ACM Special Interest Group for Design of Communications Proceedings of the 17th annual international conference on Computer documentation, ISBN 1-58113-072-4, pgs 99-104, 1999.

[JOE97] Gregor Joeris, “Change Management Needs Integrated Process and Configuration Management”, Foundations of Software Engineering, Proceedings of the 6th European conference held jointly with the 5th ACM SIGSOFT International symposium on Foundations of software engineering, 1997, pgs 125-141.

[LI02] Eldon Li, Houn-Gee Chen and Tien-Sheng Lee, “Software Process Management of Top Companies in Taiwan: A Comparative Study”, Total Quality Management, Vol 13, No 5, August 2002, pgs 701-713.

[LIP03] Walt Lipke, “Schedule is Different”, PMI CPM Journal, The Measurable News, March 2003.

[LM04] “Cost Account Managers Handbook”, Lockheed Martin, 2004.

[LOF05] Chris Loftus, Mark Ratcliffe, “Extreme programming promotes extreme learning?”, Annual Joint Conference Integrating Technology into Computer Science Education, ISBN 1-59593-024-8, pgs 311-315, 2005

[MER01] Merriam-Webster’s Collegiate Dictionary, 10th edition, ISBN 0-87779-709-9, 2001.

[NAU69] Peter Naur and Brian Randell, “Software Engineering: Report of a conference sponsored by the NATO Science Committee”, 1969.

[PAU93] Mark Paulk, Bill Curtis, Mary Chrissis and Charles Weber, “Capability Maturity Model for Software, Version 1.1SM”, CMU/SEI-93-TR-24, Software Engineering Institution, Pittsburgh, PA, February 1993.

[PRE03] Jean-Paul Prentice, “Earned Value in the Construction and Facilities Maintenance Environments”, AACE International Transactions, 2003.

[PRE04] Roger S. Pressman, SOFTWARE ENGINEERING: a Practitioner’s Approach 6th edition, ISBN 0-07-301933-X, April 2004

[REN91] Hal Render and Roy Campbell, “An Object-Oriented Model of Software Configuration Management”, Software Configuration Management Workshop, 1991, pgs 127-139.

[ZHA05] Xinpei Zhao, Keith Chan and Mingshu Li, “Applying Agent Technology to Software Process Modeling and Process-Centered Software Engineering Environment”, Proceedings of the 2005 ACM symposium on Applied computing, Santa Fe, New Mexico 2005.

BIOGRAPHICAL INFORMATION

Ricky Don Preuninger received an A.B.A. in Accounting from Grayson County College in May 1977 and a B.B.A. in Accounting from UTA in May 1980. He has been working at Lockheed Martin Missiles and Fire Control (LMMFC) for the last twenty-five years. His twenty-five years of experience at LMMFC has been primarily in Management Science, Management Information Systems, and Earned Value Management. Ricky presently holds the position of Business Systems Technology Support and Integration Manager. He plans to continue his career at LMMFC.