A Business Case for Software Process Improvement (2007 Update), Measuring Return on Investment from Software Engineering and Management

A DACS State-of-the-Art Report
September 2007

https://www.thedacs.com/

Distribution Statement A
Approved for public release: distribution is unlimited

A Business Case for Software Process Improvement

(2007 Update)

Measuring Return on Investment from Software Engineering and Management

A DACS State-of-the-Art Report

DACS Report Number 347616

Contract Number SP0700-98-D-4000
(Data & Analysis Center for Software)

30 September 2007

PREPARED FOR:

Air Force Research Laboratory
AFRL/IFT
525 Brooks Road
Griffiss AFB, NY 13441-5700

PREPARED BY:

Thomas McGibbon
Daniel Ferens
Robert L. Vienneau

ITT Advanced Engineering and Sciences
775 Daedalian Drive
Rome, NY 13441

Distribution Statement A
Approved for public release: distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE *(DD-MM-YYYY)*
30 September 2007

2. REPORT TYPE
30 September 2007

3. DATES COVERED *(From - To)*
N/A

4. TITLE AND SUBTITLE
A Business Case for Software Process Improvement (2007 Update)

Measuring Return on investment from Software Engineering And Management

5a. CONTRACT NUMBER
SP0700-98-D-4000

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
N/A

6. AUTHOR(S)
Thomas McGibbon

Daniel Ferens

Robert Vienneau

5d. PROJECT NUMBER
N/A

5e. TASK NUMBER
N/A

5f. WORK UNIT NUMBER
N/A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
ITT Advanced Engineering & Sciences, 775 Daedalian Dr., Rome, NY 13441-4909

8. PERFORMING ORGANIZATION REPORT NUMBER

DAN 347616

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Technical information Center
DTIC/AI
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060

10. SPONSOR/MONITOR'S ACRONYM(S)
DTIC
AFRL/IF

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release, Distribution Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The purpose of this revised State of the Art Report (SOAR) is to provide new insights into the details necessary to demonstrate from a business perspective the benefits of improved software management using software process improvement (SPI) techniques. SPI has received much attention in recent years; however, it has been very difficult to translate benefits achieved in one organization to another organization. The intent of this SOAR is to generalize and model the cost benefits one can achieve from SPI efforts. This revised SOAR updates the previous (1999) edition by examining the business implications of some more recent SPI practices, including the Capability Maturity Model for Integration (CMMI), agile development, and systems engineering. The Data Analysis Center for Software (DACS) has recently implemented a new capability on the DACS web site to provide updated information about return on investment (ROI results; it is the ROI Dashboard©. The ROI Dashboard© also contains updated information for practices such as inspections, reuse, and secondary benefits. A new Section has been added to this report to describe and explain this new capability. It establishes a framework whereby the current methods of performing software development can be compared to any proposed improvements.

15. SUBJECT TERMS
SOFTWARE ENGINEERING TOOLS AND TECHNIQUES, CMM, CMMI, ECONOMIC ANALYSIS,SOFTWARE ENGINEERING PROCESS, RETURN ON INVESTMENT ANALYSIS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Thomas McGibbon
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
U	U	U	UU	153	315-334-4933

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

A Business Case for Software Process Improvement (2007 Update):

Measuring Return on Investment from Software Engineering and Management

Table of Contents

1. INTRODUCTION 7

2. THE DACS ROI DASHBOARD© 11

2.1 Definition of ROI 12

2.2 Getting Started with the DACS ROI Dashboard 16
2.2.1 Box Plots 17
2.2.2 Bar Graphs 18
2.2.3 Text Display 18

2.3 Cautions and Caveats 19

3. LITERATURE REVIEW 23

3.1 Business Value of Software Management 24

3.2 Organizational Results of Process Improvement 26
3.2.1 General Organizational Results of Process Improvement 26
3.2.2 Capability Maturity Model (CMM) 28
3.2.3 Capability Maturity Model Integration (CMMI) 37
3.2.4 Personal / Team Software process (PSP/TSP) 43

3.3 Results of Specific Process 46
3.3.1 Inspections 46
3.3.2 Software Reuse 51
3.3.3 Cleanroom Software Development 56
3.3.4 Agile Development 59
3.3.5 Systems Engineering 64

3.4 Secondary Benefits of Improvement Efforts 65

3.5 Risks From Software Improvement 68

4. DETAILED RESEARCH 71

4.1 Modeling the Cost Benefit of Software Process Improvement 72

4.2 Modeling the Benefits of Specific processes 74
4.2.1 Benefits of Inspections 74
4.2.2 Modeling the Effects of Reuse 76
4.2.3 Modeling the Effects of Cleanroom Software Development 81
4.2.4 Modeling the Effects of Agile Software Development 83

4.3 Modeling Secondary Benefits of Process Improvements 86
4.3.1 Cost Benefit of Improved Schedules 86
4.3.2 Cost Benefit of Reduced Staff Turnover and Better Staff Retention 87
4.3.3 Cost Benefit of Improved Customer Satisfaction 90

4.3.4 Cost Benefit of Reduced Risk on Software Projects 91

4.4 Comparison of Results **95**

5. SUMMARY AND CONCLUSIONS **96**

5.1 The Financial Benefits of Software Process Improvement **97**

5.2 The Secondary Benefits of Software Process Improvement **99**

6. ANNOTATED BIBLIOGRAPHY **101**

APPENDIX A: INSTRUCTIONS FOR USE OF THE DACS ROI FROM SPI SPREADSHEET MODEL **147**

A.1 Introduction **147**

A.2 Software Size Estimation **148**

A.3 COCOMO Cost Estimation **150**
A.3.1 COCOMO P's Sheet 150
A.3.2 Schedules Sheet 151
A.3.3 Estimates Sheet 152

A.4 Return on Investment from Software Process Improvement **153**
A.4.1 ROI Summary Sheet 153
A.4.2 Inspections Sheet 154
A.4.3 Reuse Sheet 154
A.4.4 Cleanroom Sheet 154
A.4.5 SPI Sheet 154
A.4.6 Risks 155

1. Introduction

The purpose of this revised State of the Art Report (SOAR) is to provide new insights into the details necessary to demonstrate from a business perspective the benefits of improved software management using software process improvement (SPI) techniques. SPI has received much attention in recent years; however, it has been very difficult to translate benefits achieved in one organization to another organization. The intent of this SOAR is to generalize and model the cost benefits one can achieve from SPI efforts. This revised SOAR updates the previous (1999) edition by examining the business implications of some more recent SPI practices, including the Capability Maturity Model for Integration (CMMI), agile development, and systems engineering. The Data Analysis Center for Software (DACS) has recently implemented a new capability on the DACS web site to provide updated information about return on investment (ROI) results; it is the ROI Dashboard©. The ROI Dashboard© also contains updated information for practices such as inspections, reuse, and secondary benefits. A new Section, Section 2, has been added to this report to describe and explain this new capability. It establishes a framework whereby the current methods of performing software development can be compared to any proposed improvements. In the future, as more data on systems practices such as CMMI and systems engineering are collected, the effects on ROI for entire software-intensive systems may be included in this report.

This report emphasizes ROI because it is a very popular metric due to its versatility and simplicity, and because business managers often use it as a basis for decision making. To calculate ROI, the difference between the benefit (return) of an investment minus the cost of the investment is divided by the cost of the investment; the result is expressed as a percentage or a ratio. If an investment does not have a positive ROI, or if there are other opportunities with a higher ROI, then the investment will usually not be undertaken. A challenge for SPI efforts is that they do incur a cost of investment, and their use must be justified by a projected gain or benefit that exceeds the cost. Sometimes, they must also demonstrate a greater gain and resultant ROI than other efforts in a company or organization. This report will help the reader to do this for SPI efforts. (Section 2 of this report presents a more detailed explanation of ROI.)

More detailed information about ROI for specific SPI areas can be found on the DACS ROI Dashboard© web site, which is https://www.thedacs.com/databases/roi/. While some information about the ROI Dashboard© is presented in Section 2 of this SOAR report, detailed

instructions on using the dashboard are available by clicking on the “Overview” link at the end of the first paragraph on the Dashboard home page. (Registration is required to view ROI information. It is free; the link is https://www.thedacs.com/forms/register_step1.php.)

This report demonstrates that sound application of software engineering methods by software managers can:

♦ Increase Profitability. The ROI for software improvement is usually very high. According to van Solingen (2004), many organizations have reported an average ROI of 7:1. This high ROI is achieved by reducing development costs, rework costs, and turnover costs. Product sales increase from higher quality software, penalties turn into bonuses, and repeat business increases. Furthermore, a risk analysis of performing software improvements versus not performing the improvements highly favors performing the improvements.

♦ Reduce software development and maintenance (or support) costs. The cost of implementing SPI methods and practices are usually heavily outweighed by the cost savings from reduced development costs, and the cost savings resulting from less rework. The major reduction of development costs can be attributed to improved software productivity. Implementing SPI methods also reduces maintenance costs, which often exceed development costs.

♦ Reduce Cycle Time. Improvement efforts can reduce typical development schedule lengths by 30% to 40%. This may translate to higher profit because it may allow organizations to beat the competition in getting a product to the field, result in more products purchased earlier than projected, or result in schedule-related bonuses for early delivery. Combining improved schedules with higher quality, getting better products out sooner, is a winning combination as far as customers are concerned.

♦ Improve quality and customer satisfaction. Typical software development organizations release products with 15% of the defects remaining for the customer to find. No customer is happy with that many problems. Some SPI methods can reduce post-release defects to near zero. Improving customer satisfaction is shown to result in repeat customer business and an improved company image.

♦ Improve Professional Staff. SPI improves employee morale, and increases the confidence of developers. It results in less overtime, fewer crises, less employee turnover, and an improved competitive edge. The reduction in employee turnover costs and retraining costs can pay for the improvement costs alone.

The focus of this report is on SPI and the cost benefits that can be achieved through an SPI program. This paper examines overall organizational results from SPI and the results from specific SPI practices. (This is consistent with the improvement area matrix on the ROI Dashboard©.) The organizational results include the Capability Maturity Model (CMM), the Capability Maturity Model Integration (CMMI), and usage of the Personal and Team Software Processes (PSP/TSP). Specific practice results include inspections, software reuse, Cleanroom software development, agile development, and systems engineering. This report also examines some of the general and detailed benefits, secondary benefits, and risks associated with software engineering initiatives.

Section 2 of this report, added to this revised edition, explains ROI and the contents and features of the DACS ROI Dashboard©. The information available from the ROI Dashboard© can further explain the information contained in this report, provide the latest up-to-date data, and provide additional justification information for SPI efforts.

The literature has many documented success stories of cost savings resulting from software process improvements. Section 3 of this report summarizes the findings as reported in the literature. This Section also explores other writers' views of the business benefits or value of process improvement and software management as well as literature that discusses the secondary benefits of process improvement. There are many secondary benefits discussed in the literature, including higher customer satisfaction, improved employee morale, and an improved competitive position which are also discussed in Section 3.

In Section 4 of this report, for each of four classes of SPI (inspections, software reuse, Cleanroom methodology, and agile development), a spreadsheet model is developed which will allow anyone to identify likely cost savings from SPI in his or her organization. This spreadsheet is built upon the Constructive Cost Model (COCOMO), a software cost and schedule estimating model developed by Dr. Barry Boehm (1981). An enhanced spreadsheet is also provided to provide a framework for addressing the secondary benefits of improvements.

In Section 5 of this report, generalizations about the spreadsheet of Section 4 are made and conclusions are drawn from the research performed. Section 6 includes an annotated bibliography.

We recognize that methods other than SPI can have significant improvements and cost savings on a software development organization. For example, Boehm (1987) has shown that

utilizing a second-rate (15th percentile) team on a software project requires 4 times (4X) as many man months of effort as using a highly skilled (90th percentile) team. Boehm (1987) has also shown that investments in programmers' facilities have been recaptured with improved productivity. However, to consistently achieve quality software products utilizing average skilled teams, an SPI program is needed.

We also recognize that some people have not believed in the validity of the ROI data in the literature. For example, Fenton (1993) provides a skeptical view of the statistics and ROI data reported in the literature. He shows that anecdotal "evidence" of significantly improved quality and productivity are not backed up by hard empirical data, and where hard data exists, it is counter to the view of the "so-called" experts. More recent literature, however, shows that data is usually valid and SPI usually does result in improvements.

Appendix A contains instructions for using the attached diskette titled "The DACS Return-On-Investment (ROI) from Software-Process-Improvement (SPI) Spreadsheet Model," containing the Microsoft Excel® spreadsheet which can be used for software size estimation, software cost estimation, and ROI from SPI analysis.

2. The DACS ROI Dashboard©

The DACS has continued to collect ROI data from open source literature for more than a decade after the first edition of this report was issued. Much of the quantitative reported improvements for SPI and specific processes are measured against a small number of project attributes, such as cost, schedule, productivity, and quality. DACS efforts have captured, codified, and represented these improvements in a common database format. The DACS continues to update this database as new data emerges in the open literature.

The DACS *ROI Dashboard*© takes the next step by providing a web-based set of graphical and analytical tools, interfaced to this database. These tools provide ways for users visualize and reason about the reported data. The *ROI Dashboard*© is available through the DACS website at http://www.thedacs.com/databases/roi/. The DACS website requires registration, which is free.

This Section summarizes a few of the key capabilities available from the *ROI Dashboard*©. Figure 2.1 presents the *ROI Dashboard*© home page. From this page, DACS users can select improvements of interest and how they would like the costs and benefits of such improvements displayed. The *ROI Dashboard*© provides a valuable visual and analytical tool for the engineer or manager, wishing to rationalize or justify investments in improvement, to quickly assess the extent of improvements that have occurred within other organizations. This tool provides still other capabilities and information on other improvements than described in this introduction.

If you have observed results data for any improvement and would like to include them in the *Dashboard*, please contact the authors at the DACS, or, better yet, complete the SPI data collection survey form accessible by clicking 'Submit a Case Study' on the ROI Dashboard home page (see http://www.thedacs.com/databases/roi/submit/). Either way, the DACS will contact you to follow up on the submittal.

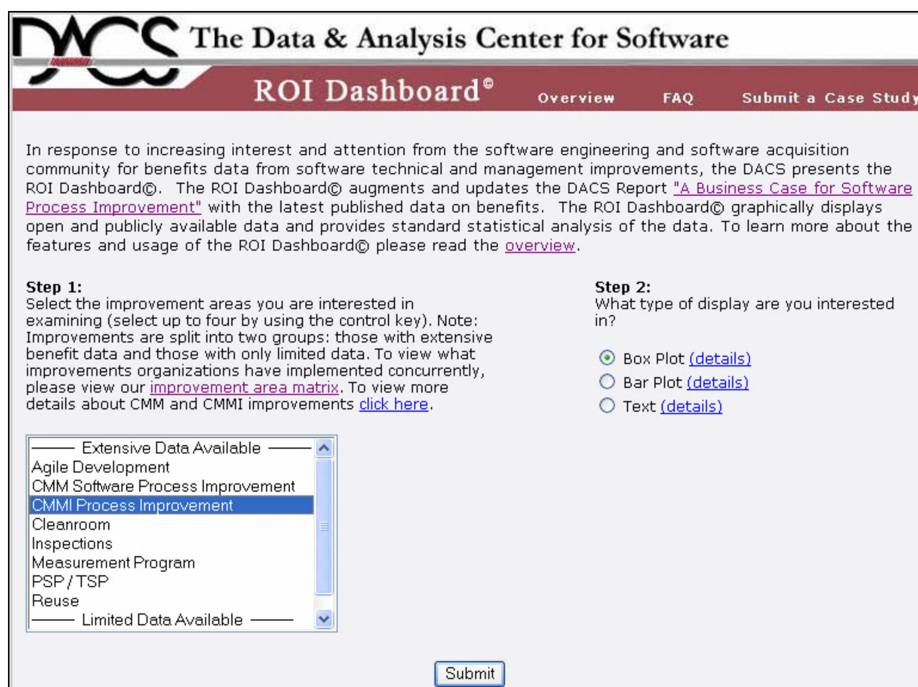

Figure 2.1: Selecting Improvements and Display Type from the *ROI Dashboard*©

2.1 Definition of ROI

This document and the *ROI Dashboard* report the business benefits of software improvements in terms of ROI. ROI, as used in financial analysis, has a precise mathematical definition. The term's usage has also expanded among management. It is a metaphor for benefits received as a result of incurring some costs. Often this metaphoric usage remains quantitative. In this report, ROI is used in both the financial analysis sense and as a metaphor. Quantitative benefits from SPI and specific methods are measured in terms of ROI under the financial analysis definition, productivity, lower project cost, lower cycle times, improved quality, decreased rework, and smaller schedule variance.

ROI, in a precise sense, is the internal rate of return of an investment. An investment, most generally, can be considered a stream of payments or revenues. One pays into an investment for prospective revenues returned at specified dates in the future. Consider an investment that requires payments in or produces revenue at the start of each year. Let y_t ; $t = 0$, 1, 2, …, n; be the revenue produced at the start of the nth year. A payment, that is, a cost, is represented by a negative value of y_t. Typically, y_0, the revenue at the start of the first year of the investment, is negative. The internal rate of return is a rate of interest, r, for which the net

present value of the investment is zero; that is, the internal rate of return solves the following equation:

$$0 = y_0 + \frac{y_1}{1+r} + \frac{y_2}{(1+r)^2} + \cdots + \frac{y_n}{(1+r)^n}$$

Frequently, a simplified formula is used for ROI. Consider an investment that requires one payment at the start and yields revenue at exactly one point in time. Let V be the value of the original investment and let Y be the return at the end of the period. In terms of the above definition, the ROI is:

$$ROI = \frac{Y - V}{V}$$

In summarizing the literature, we use the above formula and ignore the precise time pattern of payments and yields. In some documents in the literature, it is challenging to determine whether the returns described are with or without the original investment subtracted. We interpreted this ambiguity in specific articles as best we could.

More broadly, ROI is the improvement gained by the expenditure of some cost. Improvements described in the literature include increased productivity, lower project cost, decreased cycle time, increased quality, decreased rework, and less variance between actual costs and schedules and budgeted costs and schedules. The *Dashboard* accommodates disparities in units of measure. For example, project sizes can be measured in Source Lines Of Code (SLOC) or Function Points. Cost might be measured in dollars or person-months. Cycle time need not be converted a single time measure like weeks, months, or years. The *Dashboard* also accommodates concerns organizations may have about company-proprietary data; for example, some organizations may not want to publish productivity or failure rates. These accommodations are made by recording percent improvements when raw data is not available and by separately recording units of measurement for raw numbers.

Percent improvements are dimensionless. For example, suppose P_B is the productivity of some organization's software process before implementing some improvement, such as achieving assessment at a higher CMM level. Let P_A be the productivity after implementing the improvement. Then, the percent increase in productivity is:

$$100\frac{P_A - P_B}{P_B}$$

One can report this percent increase without reporting the productivities before and after the improvement. The productivities before and after the improvements must be in the same measurement units for a single data point. Percent increases in productivity, however, can be compared across data points even if the raw productivities are measured in different units.

Measures of percentage improvements (not ROI by a strict finance definition) are calculated based on measures of:

- **Productivity:** The amount of output produced per unit input. Software productivity is typically assessed in SLOC per staff-day or Function Points per person-month, for example. The *Dashboard* graphs percent increase.
- **Quality:** Two measures of quality are used in the *Dashboard*:
  - **Defect density:** In units such as delivered defects per thousand SLOC or defects found during acceptance testing per thousand Function Points. The *Dashboard* graphs percent decrease.
  - **Percentage of defects found:** In units such as percent of defects found by a specified activity or the percent of total defects found before delivery. The *Dashboard* graphs percent increase.
- **Rework:** The cost, often measured in the ratio of time or dollars to overall project time or dollars, required to alter a component or product to correct a defect. The *Dashboard* graphs percent decrease.
- **Project Cost:** The total amount spent on a project. The *Dashboard* graphs percent decrease.
- **Cost of the Improvement:** The total cost required to implement a process improvement. This usually includes the training cost and the cost to perform the new process (as in formal inspections).
- **Cycle Time:** The time from inception to product release; the total schedule length for a software development project or iteration. The *Dashboard* graphs percent decrease.
- **Schedule Variance:** See following. The *Dashboard* graphs percent decrease.

Schedule Variance is a measurement used in Earned Value Analysis (EVA), a project management methodology to measure progress objectively. The amount of work performed by a given date is measured by the budget for the products produced by that date is the Planned Value (PV), while the amount of work scheduled by that date is measured by the expenditure budgeted to be spent by that date is the Earned Value (EV). Schedule Variance is the difference between EV and PV. Schedule Variance is negative for a project that runs over schedule and positive for a work completed ahead of schedule.

The distributions of ROI and percent changes for SPI and specific improvements are presented by graphical displays highlighting certain statistics. These statistics are measures of central tendencies and measures of how the data is dispersed about the central tendency:

- **Minimum:** The smallest value in the sample.
- **Maximum:** The largest value in the sample.
- **Mean:** A measure of central tendency. The mean is the quotient of the sum of the values divided by the number of elements in the sample.
- **Median:** Another measure of central tendency. The median is such that half the sample is less than its value. The median and the mean are equal for Gaussian (bell-shaped) distributions. The median is less sensitive to outliers and extreme values than the mean.
- **Standard deviation:** A measure of dispersion. The square of the standard deviation is the quotient of a certain sum and one less than the number of elements of a sample. That sum is the sum of the squares of the difference between the value of each element of the sample and the mean. The standard deviation is in the same units of measure as the elements of the sample.

  For Gaussian (bell-shaped) distributions, 68.3% of the population is within one standard deviation of the mean, 95.4% is within two standard deviations, and 99.7% is within three standard deviations.
- **25th Percentile:** Also known as the first quartile. The first quartile is such that one quarter of the sample is less than its value.
- **75th Percentile:** Also known as the third quartile. The third quartile is such that three quarters of the sample is less than its value.

2.2 Getting Started with the DACS ROI Dashboard

The *Dashboard* home page (Figure 2.1) presents the user with a choice of process improvements and a choice of displays. In the input window on the left, the user selects one or more improvement areas of interest. On the right, the user chooses his or her preferred display type (Box Plot, Bar Plot, or Text Display), and then clicks the "Submit" button. The resulting data will be presented in the chosen format. Figure 2.2 provides a visual comparison of a box plot and bar plot of the same data, while Figure 2.3 provides a sample text display. All three graph types allow the user to drill down to the detailed data and sources, simply by clicking on the box, bar, line, point, or the links embedded within the text display. The following subsections provide a detailed explanation of each of the three presentation formats.

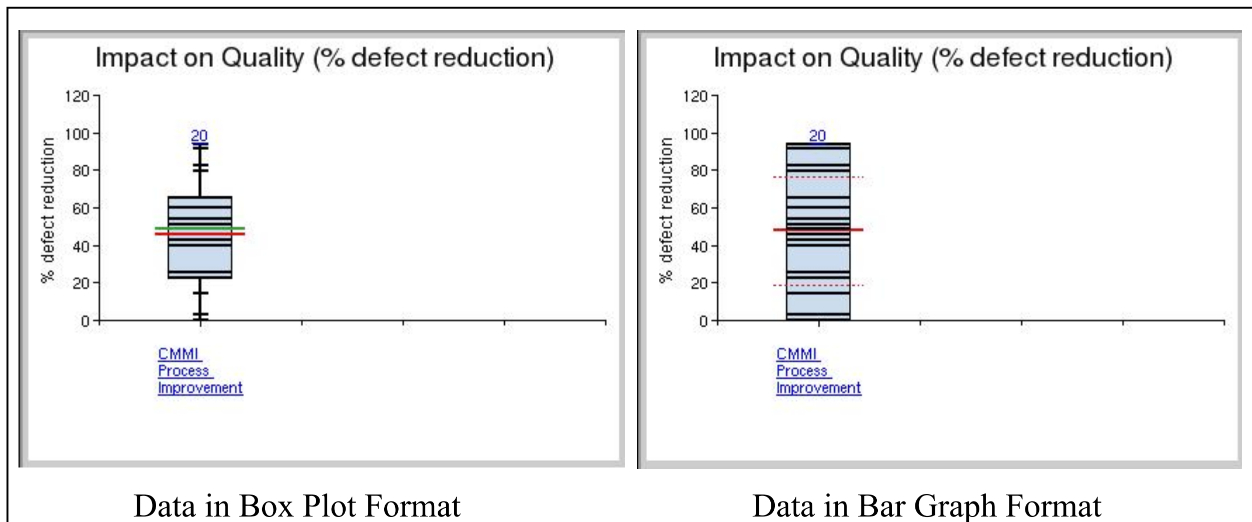

Figure 2.2: Sample Box and Bar Plot for the Same Data

Metric	Total Data Points	Minimum	Maximum	Median	Mean	Standard Deviation	25th Percentile	75th Percentile
ROI	9	2 Ratio	13.3 Ratio	3 Ratio	4.64 Ratio	3.55 Ratio	2.25 Ratio	5.5 Ratio
Impact on Cycle Time	5	15 % decrease	50 % decrease	38 % decrease	32.6 % decrease	14.62 % decrease	17.5 % decrease	45 % decrease
Reduction in Rework	1	60 % decrease	60 % decrease	60 % decrease	60 % decrease	0 % decrease	0 % decrease	0 % decrease
Impact on Quality (% defect reduction)	20	0.5 % defect reduction	95 % defect reduction	48.5 % defect reduction	47.64 % defect reduction	29.21 % defect reduction	25.5 % defect reduction	67 % defect reduction
Impact on Productivity	12	5 % improvement	250 % improvement	39 % improvement	57 % improvement	67.5 % improvement	13.5 % improvement	66.5 % improvement
Impact on Schedule Variance	3	35 % decrease	50 % decrease	40 % decrease	41.67 % decrease	7.64 % decrease	35 % decrease	50 % decrease
Impact on Quality (% of defects found)	1	98 % defects found	98 % defects found	98 % defects found	98 % defects found	0 % defects found	0 % defects found	0 % defects found
Reduction in Project Cost	2	20 % decrease	40 % decrease	30 % decrease	30 % decrease	14.14 % decrease	20 % decrease	40 % decrease
Cost of the Improvement	1	1.1 % of total engineering effort	1.1 % of total engineering effort	1.1 % of total engineering effort	1.1 % of total engineering effort	0 % of total engineering effort	0 % of total engineering effort	0 % of total engineering effort

Figure 2.3: Sample Text Display

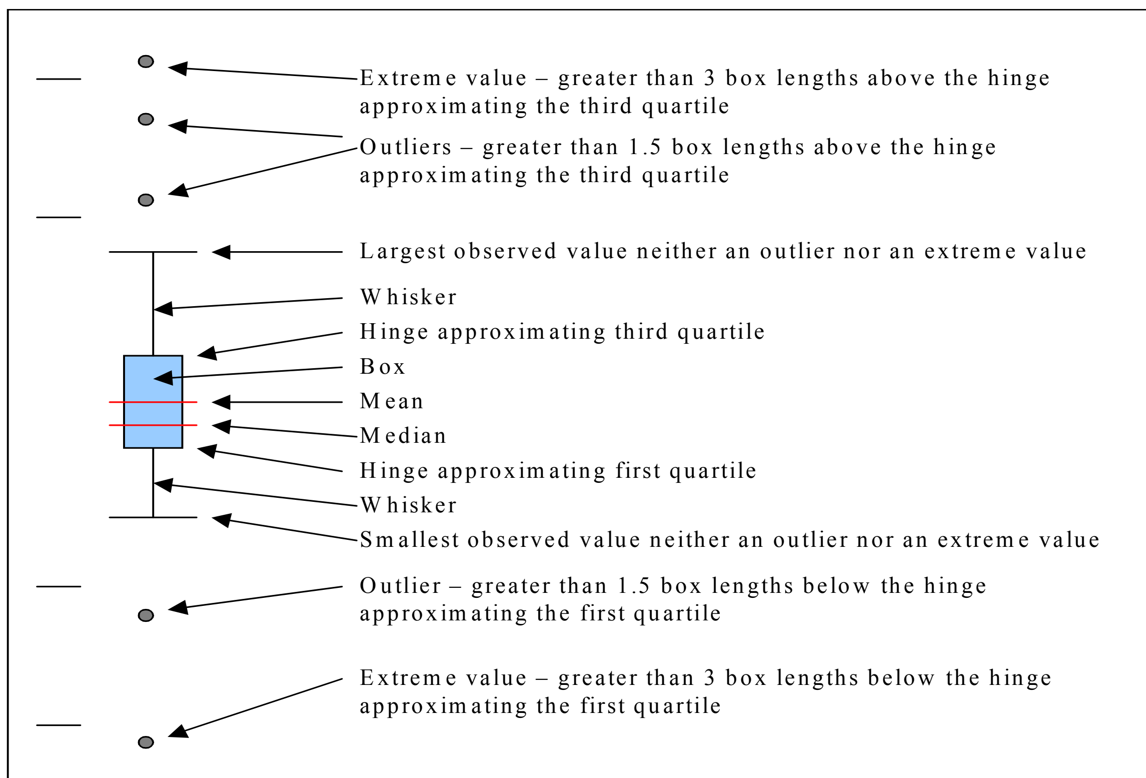

Figure 2.4: Defining a Box Plot

2.2.1 Box Plots

Figure 2.4 presents a graphical definition of a Box Plot, also called a box and whiskers plot. A Box Plot displays the empirical distribution of a single variable (such as impact on quality). Half of the distribution is in the center box. Whiskers, at the top and bottom of the box, show the extent of most of the remainder of the distribution. Finally, outliers and extreme values are plotted beyond the whiskers. Tukey (1977) first described Box Plots.

The Box Plot shows various statistics. Box Plots on the DACS *ROI Dashboard*© display two measures of central tendency, the mean and the median. Traditional Box Plots do not display the mean. The median is less sensitive to extreme values and outliers; it is also easier to interpret for non-Gaussian (non-bell-shaped) and non-symmetric distributions.

The lower and upper edges of the box, known as "hinges", approximate the first and third quartiles of the distribution. (The median is the second quartile.) The lower hinge is the median of the points less than or equal to the median. In cases where the first quartile is found by interpolating between two data points, the hinge will typically come out as a different interpolation. (Tukey defined hinges for ease of calculation.) The upper edge is found, similarly, as the median of the points greater than or equal to the median. The upper hinge approximates

the third quartile. Note that the central half of the distribution is between the two quartiles. The inter-quartile range is a measure of the variability of the data.

2.2.2 Bar Graphs

Figure 2.5 presents the defining characteristics of a Bar Graph. The statistics shown on a Bar Graph consist of the mean and multiples of the standard deviation. If the distribution were Gaussian or bell shaped, most of the distribution would be within a couple of standard deviations of the mean. For a Bar Graph, where the bulk of the distribution lies is shown by horizontal black lines (not shown in Figure 2.5) representing individual data points. In many ways, the Box Plot presents a more articulated picture of the distribution of data than a Bar Graph does. The Bar Graph, however, is useful for visualizing standard deviations.

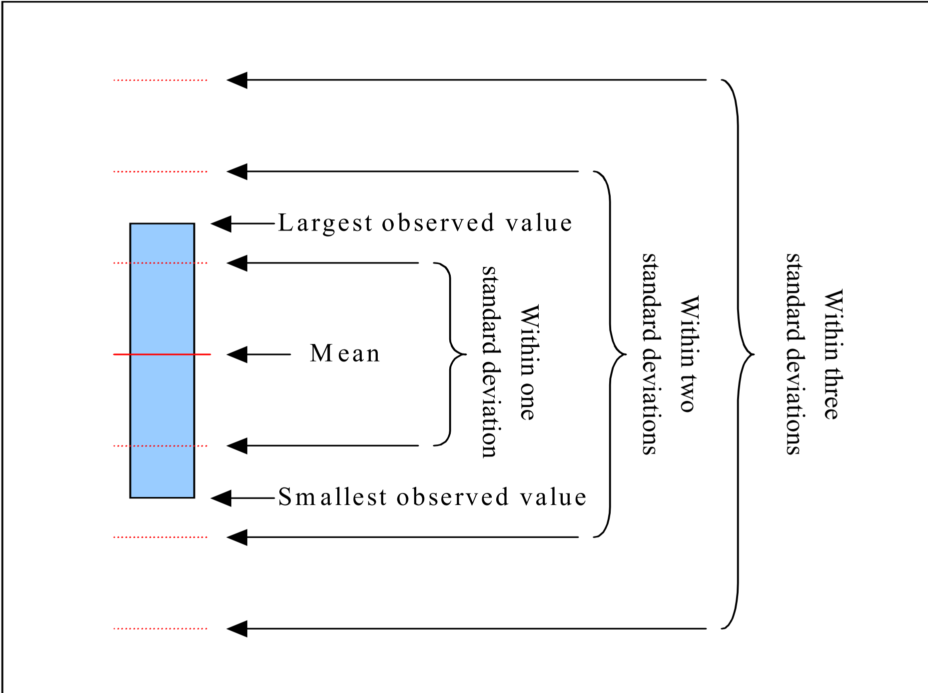

Figure 2.5: Defining a Bar Graph

2.2.3 Text Display

The Text Display, illustrated in Figure 2.3, provides various distribution-free and parametric statistics about each attribute (metric) of the improvement area for which data exists. These statistics consist of the total data points, minimum, maximum, median, mean, standard

deviation, 25th percentile, and 75th percentile. The user can click on the hyperlinked metric name or the total data points to obtain details about data and data sources.

2.3 Cautions and Caveats

The *Dashboard* provides an overview of quantitative data reported in the literature on software process improvements, including some specific improvements. Table 2.1 shows DACS users who have generated graphs and reports from the *Dashboard* since we have required users to register. (Users who revisited the *Dashboard* in more than one month are only counted once in the last column; thus, for some rows, the entry in the last column is less than the sum of monthly columns.) This data suggests practitioners find the data in *Dashboard* more worth exploring than academics (professors and students) do. Furthermore, 61% (94 out of 154) of the users are in management. An overview of quantitative business benefits of software process improvements seems to be of interest to executives, managers, and their staff.

Role	**May 2007**	**Jun 2007**	**Jul 2007**	**Aug 2007**	**All**
President	0	1	0	0	1
Vice President	1	0	1	0	2
Metrics Mgr and 6 Sigma MBB	0	1	1	0	2
Senior Mgmt	3	3	11	2	18
Program/Project/Eng. Mgr.	5	8	24	21	53
Acquisition Program Analyst	0	0	1	0	1
Research and Development	0	4	2	1	6
Consultant	0	0	1	0	1
SQA/SQE Mgr.	3	6	6	3	17
Software Test Engineering Mgr.	1	0	0	0	1
Software/Systems/Program Analyst/Engineer	3	4	8	7	20
SQA Analyst/Engineer	3	4	4	3	14
Academia/Professor	0	2	0	0	2
Student	0	0	0	1	1
Miscellaneous and Unknown	1	4	5	5	15
Total	20	37	64	43	154

Table 2.1: Users of the *ROI Dashboard*© by Role and Month

The *Dashboard* methodology accommodates the range of statistical rigor reported in software engineering literature. Some literature reports on controlled experiments, justifying

statistical hypothesis testing of the effects of the “treatment”, that is, a specific improvement in software technology or process. Other literature is more observational, with values without the treatment being drawn from organizational or industry averages. Some papers report data from single projects, while others report data from a single organization or from surveys of many organizations. Not all the studies in such literature can be quantitatively combined or aggregated using rigorous statistical techniques, described as meta-analyses.

Social science and medical research are probably the most common application areas for meta-analysis. Hayes (1999) and Miller (2000) illustrate meta-analysis in software engineering with small case studies. Effect sizes are typically reported in meta-analyses on a different scale than that used in the *Dashboard*. A typical measure is the difference between the control and treatment groups, normalized by some estimate of the standard deviation. Other measures include statistical significance levels and odds ratios. Deviations in the odds ratio from unity indicate how much more or less likely an effect is to be observed in the treatment or control group. These measures allow the meta-analyst to calculate how much random variation would be expected in measures of effect sizes among the individual studies if the treatment had no effect. These calculations account for differences in sample sizes among the studies. Some approaches account for variability in quality and heterogeneity in measures of effects across the studies. If the observed random variation among the studies is greater than that to be expected if the treatment were ineffective, the meta-analyst can reject the null hypothesis of no effect at a controlled statistical significance level. One can also estimate a confidence bound for the overall effect size and test whether the populations sampled by the individual studies vary among them.

Most of the literature summarized in the *Dashboard* lacks characteristics needed to support these meta-analytical techniques. Many of the studies are not controlled experiments, and sample sizes, standard deviations, and statistical significance levels are frequently not reported. The *Dashboard* methodology of reporting effect sizes in terms of percent improvement allows visualization of the data taken from this wider range of studies than can be accommodated in a statistically rigorous meta-analysis. How much further one would want to push the analysis is a matter of judgment for the *Dashboard* user. Galin and Avrahami (2005), for example, statistically test the impact of transitioning from one CMM level to another based on data also reported in the *Dashboard*. As a minimum, the Dashboard provides a guide to literature on quantitative assessments of software process improvements.

The *Dashboard* does not capture all differences in business aspects of process improvements. In particular, the scale of investments and the quantities of benefits are not captured in a financial measure of ROI. For example, implementing formal inspections or peer reviews on a single project will require a smaller investment than successfully completing an organization-wide CMM or CMMI assessment. The total return for implementing formal inspections will, however, typically be smaller than for CMM SPI, even though the mean ROI, expressed as a percentage, for formal inspections exceeds the mean ROIs for CMM and CMMI.

The *Dashboard* user should be aware that data is aggregated into the improvements in the database. The data for some process areas contains literature on diverse methodologies. For example, data on Agile Programming includes eXtreme Programming (XP), the Dynamic Systems Development Method (DSDM), and other methodologies. Sources of data on CMM include, for example, Raytheon Software Systems Laboratory transitioning from the Initial Level to the Defined Level, as well as Motorola transitioning from the Repeatable Level to the Optimizing Level. Perhaps ROI varies among agile methodologies or depends on exactly from and to which CMM level an organization transitions. Another issue arises in decomposing the benefits shown in the literature to the different elements that comprise a process. Often such decomposition is not possible with the *Dashboard* data, although some of the literature is suggestive. For example, many of the controlled experiments on XP focus on pair programming, only one of the XP practices. As another example, the implementation of peer reviews, a process with high ROI, is just one of thirteen Key Process Areas required to achieve the CMM Defined Level. When an article reports an organization as implementing more than one improvement, without decomposing ROI data by improvement, the *Dashboard* reports that data for both improvements. Despite these issues, some level of aggregation is needed for the current state of the literature to obtain sample sizes large enough to exhibit statistical variation. Even so, the DACS was only able to locate limited data for some improvement areas.

The source of the data in the *Dashboard* is open literature, concentrating empirical software engineering and process improvement. Journals in which articles were found include *Communications of the ACM*, *CrossTalk*, *IBM Systems Journal*, *IEEE Software*, *IEEE Transactions on Software Engineering*, *Journal of Systems and Software*, *Software Practice and Experience*, and *Software Process – Improvement and Practice*. Conferences in which articles with SPI ROI data were found include conferences sponsored by the Software Engineering Institute; Software Engineering Process Group (SEPG) conferences; and conferences dedicated

to specific improvements, such as agile programming. Some data came from a DACS survey, and some from textbooks, where appropriate.

3. Literature Review

This report presents a business case for performing software process improvement. The primary emphasis of the literature in this area focuses on benefits achieved from increased productivity and decreased rework - the primary benefits. Literature addressing this subject area discusses such elements as cost savings from process improvements, ROI, improved productivity, and improved cycle times[1] from process improvements. The literature in this area is divided into five categories:

- Literature that discusses the business value of software engineering and engineering management
- Literature that presents results from organizations that have invested heavily in SPI using processes such as the Capability Maturity Model (CMM), the Capability Maturity Model for Integration (CMMI), and Personal Software process / Team Software process (PSP/TSP), and shows the collective benefits, risks, and lessons learned from applying software management techniques.
- Literature that addresses specific process improvements that resulted in significant savings or benefits to organizations. Two specific process improvement areas dominate the literature: software inspections, and software reuse. Three other specific areas, Cleanroom software development, agile methods, and systems engineering, have not received as much attention in the literature, but have shown some impressive results in cost savings and quality improvements over the entire product life cycle.
- Literature that indicates there are also many secondary benefits and other factors which need to be considered in examining and measuring software improvement from a return on investment perspective.
- Literature that discusses risks and other factors which need to be considered.

[1] Cycle time is defined as the time from the definition of product requirements to release to the customer.

3.1 Business Value of Software Management

The first class of literature relates to articles that discuss how to measure the value or return on investment from applying software engineering or software management principles, such as SPI. Managers are usually required to show that their company's investment in SPI is profitable and is in line with their business strategy.

Johnson (1997) described the Cohen Act of 1996 (the Clinger-Cohen Act, Division E of the FY96 Defense Authorization Act, Public Law 104-106), in which Information Technology (IT) is defined as:

"Any equipment, or interconnected system or subsystem of equipment, that is used in the automatic acquisition, storage, manipulation, management, movement, control, display, switching, interchange, transmission, or reception of data or information by the executive agency. It includes computers, ancillary equipment, software, firmware and similar procedures, services (including support services), and related resources. It does not include any equipment that is acquired by a Federal contractor incidental to a Federal contract."

The Cohen Act defines a set of acquisition and management practices needed to build the IT infrastructure outlined in the 1993 National Performance Review (NPR). The 1993 NPR concluded that investments in IT within the United States will make it possible to reduce waste, increase efficiency, improve customer satisfaction, and lower costs. Wise adaptation of IT, as noted in the article, can substantially improve mission performance and reduce costs. However, poor management practices, especially in the acquisition of software, have caused agencies to fail to reap the benefits of IT.

Strassman (1990) examined the value of IT to business. In his book, he examines many studies and concludes that very little is known about measuring the value of information technologies, regardless of any measure of excellence. He believes there is no known correlation of the value of IT to business because IT is not an independent variable. He concludes that information technology is not a direct cause of profitability, but a contributor to profitability. The author derived a new value-added metric, Return-on-Management (ROM), which is heavily correlated to the profit companies make. He argues that ROM is a more suitable measure than Return on Investment (ROI) or Return on Assets (ROA) in evaluating investments in IT because ROM focuses on the productivity of management. The author also noted that the business value of IT is the present worth of gains reflected in business plans when you add IT, which equals the difference of the business plan when you add IT and the business plan without changes to IT.

The author believes risk analysis of IT investments is a very important aspect of IT selections. According to Strassman, "Risk analysis is the correct analytical technique with which one can examine the uncertainty of Information Technology investments prior to implementation." He believes that "By making the risks of technology more explicit, you create a framework for diagnosing, understanding and containing the inherent difficulties associated technological and organizational innovation." Because of this argument, some of the metrics proposed in this paper are based on risk analysis techniques. He further suggests other measurements of business value from IT: gains in market share, better prices, reduced inventories, or highly motivated employees.

ROI measures for evaluating IT are discussed by Violino (1997). Violino stated that management often has difficulty computing ROI from IT. The author polled 100 IT managers to understand the importance of ROI calculations in IT investments in their organization. Of those polled, 45% require ROI calculation, 80% say ROI is useful, only 20% have formal ROI measures, and 25% have plans to adopt ROI measures in the next 12 months. He believes that some new "intangible" ROI measures are starting to appear such as product quality off the assembly line, customer satisfaction after an interaction, and faster time to market. The author contends that these measures reflect a company's real sources of value and are what customers truly care about. According to Violino, the timing of investing in IT is another intangible IT ROI factor.

Brynjolfsson (1993) examined why there is such a shortfall of evidence about productivity increases from Information Technology. Since the 1970s, when corporate investments began in technology, productivity for the production sector has increased; however, productivity for the service sector has decreased with investments in IT. In this article he addresses four possible explanations for this phenomenon: mismeasurement of outputs and inputs, lags due to learning and adjustment, redistribution and dissipation of profits where IT may only benefit certain areas (where IT rearranges the shares without making it any bigger), and mismanagement of information and technology. The author believes the major problem is due to mismeasurement. For example, if a company decides to offer many varieties of a product, its productivity appears to be lower than a company that only offers one kind.

Within the software process improvement community, McGarry and Jeletic (1993) have identified five steps that are necessary to determine the benefits of process improvement: (1) set goals for what is to be improved, (2) establish a basic understanding of an organizations current software process and product, (3) invest in change, (4) measure the effects of the change to

determine if any improvement has been achieved, and (5) measure the ROI by (a) determining what resources have been expended, (b) establishing what improvements, both qualitative and quantitative, have been achieved, and (c) determining the difference between the investment made and the benefits obtained.

Harrison, et al. (1999) discussed financial measures that may be used to make a business case for SPI. Present value methods such as net present value, internal rate of return, and profitability indexes can be useful in assessing the benefits of SPI efforts. Risk assessment should be considered in computations.

Reifer (2002) showed that building an effective business case for SPI must be based on quantitative assessments. Justification must be based on sound business rationale and not on technology alone. Reifer describes the various tools for business case analysis, such as cost benefit analysis and investment opportunity analysis, and includes several case studies to show how these tools can be used for SPI analysis and determining ROI.

Tockey (2005) discussed a multitude of factors that must be considered in making technical software decisions based on business considerations. The ROI for software activities, including SPI, must consider factors such as interest, inflation, depreciation, cost accounting, income taxes, and estimating uncertainty. Like Reifer, Tockey emphasizes the need to align activities such as SPI with business objectives.

In summary, business considerations will determine whether a company or organization will determine whether to invest in SPI and how much to invest. The remainder of this paper will show that it is usually quite profitable to invest in SPI; however, the amount invested and the method or methods selected will vary with the needs of the company or organization.

3.2 Organizational Results of Process Improvement

Process improvement efforts have provided immense benefits to many organizations. Some of the general benefits to organizations are now discussed, and some benefits from more particular areas, such as using the CMMI, are now explained.

3.2.1 General Organizational Results of Process Improvement

Several organizations have reported overall results of using SPI without referring to a specific area such as CMM or PSP/TSP. For example, Dion (1993) reported a 7.7:1 return on investment ($580 Thousand invested versus $4.48 Million saved in rework costs) and a 2 times

(2X) increase (130% per year for 4.5 years) in productivity from Raytheon's SPI efforts. Raytheon focused on development of, and compliance with, the software engineering development policies and procedures, training of engineers in the development methodology, application of advanced software development and process tools, use of formal inspections and the creation of a process (metrics) database. Raytheon computed the benefit of improvements by differentiating the costs into the categories of doing it right the first time versus the cost of rework. Based on their SPI, Raytheon has eliminated $15.8 million in rework in less than 5 years (41% of project costs before SPI program versus 11% after the SPI program). Other benefits resulting from their SPI program are that employees feel the company wants them to do a good job, higher employee morale, less absenteeism, lower attrition rates, and fewer nights and weekends required by employees. Raytheon has won two new projects and has earned a $9.6 million schedule incentive bonus because of their SPI program.

Curtis (1995) concluded that software process improvement works with a measured ROI of 6:1, a 2X or 3X productivity improvement and nearly a 100X reduction in post release defects. He points out that it is difficult to measure cost benefits from process improvements in immature organizations because immature organizations rarely have good cost data. He claims that the first benefit resulting from SPI is the ability to meet schedule. For example, Schlumberger improved on-time delivery from approximately 50% of its projects to 99% of its projects in less than three years through process definition and control, and improved project planning and control. By use of software quality assurance, post release defects also dropped from 25% of total defects to 10% in less than three years. Through use of the CMM, Hughes has learned that its cost estimates are more credible in negotiations, the effect of changing requirements is predictable, and there is less overtime and fewer crises in the software organization.

As shown in Table 3.1, Jones (2000) documented the per employee cost of software process improvement and identifies seven stages through which an organization moves on its way to maturity. During baseline assessments and benchmarking, which Jones calls "Stage 0", organizations perform a formal process assessment and establish a quantitative baseline of current productivity and quality levels. In Stage 1, software managers are trained in planning, sizing, estimating, tracking, measurement, and risk analysis. Stage 2 concentrates on the software development processes to be followed. Stage 3 is acquisition of improved tools and exploration of new technologies. Stage 4 addresses the organization and infrastructure of the organization. During stage 5, an effective reuse program is established. Stage 6, the final stage,

involves achieving leadership, through acquisitions, in a chosen specialization. The range of per employee costs to achieve each of these stages is a function of the company size, with larger companies incurring greater costs.

Stage	Focus at This Stage	Minimum Cost of SPI/Employee	Maximum Cost of SPI/Employee
0	Baseline Assessments	$100	$250
1	Management Methods	$1,500	$5,000
2	Software Process & Methodologies	$1,500	$4,500
3	New Tools & Approaches	$3,000	$8,000
4	Infrastructure/Specialization	$1,000	$6,500
5	Reusability	$500	$6,000
6	Industry Leadership	$1,500	$4,500
	Total SPI	$9,100	$34,750

Table 3.1: Costs of Software Process Improvement (Jones, 2000)

Depending on the size of the company, Jones believes improvement can take between 26 calendar months for companies of fewer than 100 people and 83 calendar months for companies of more than 10,000 people with an ROI range of 3:1 to 30:1. SPI can result in a 90% reduction in software defects, a 350% productivity gain and a 70% schedule reduction. The largest ROI does not occur until Stage 5 is attained.

3.2.2 Capability Maturity Model (CMM)

During the late 1980s and early 1990s, The Software Engineering Institute (SEI) at Carnegie Mellon University developed and, later, enhanced a five-level evolutionary process model of the capabilities of software development organizations called the Capability Maturity Model (CMM). According to this model, described by Humphrey (1989), organizations begin at a chaotic initial level and then progress through repeatable, defined, managed, and, finally, optimizing levels. Except for the initial level, each level of the model has defined key process areas (KPAs) that identify those areas on which the organization must focus on to raise its software process to that level. The CMM was finalized in 1991 by the SEI, and a number of reports and papers have been written since then which identify the costs and payoffs from process

improvement employing this model. Hayes (1995) has observed that moving from level 1 to level 2 of the CMM requires, on average, 30 months, and moving from level 2 to level 3 requires 25 months.

Humphrey (1991) was one of the first to publish benefits from using the SPI method he helped to develop. He reported on results of SPI at Hughes Aircraft. Hughes concentrated on improvements in quantitative process management, process group formation, software quality assurance, training, and reviews. Hughes has saved $2 million annually from these improvements on an investment of $445,000. Hughes has noticed an improved quality of work life with fewer overtime hours and less employee turnover. The company's image has also been enhanced because of these improvements.

At Tinker Air Force Base (AFB), Lipke (1992) reported a ROI of 6.35:1 ($462,100 invested and $2.935 million returned) from improvements recommended in their first CMM appraisal. Lipke believes that the necessary ingredients for success in SPI are leadership by senior management, recognition from everyone that process improvement is their job, and visible progress.

Wohlwend (1993) reported results from the SPI programs at Schlumberger, an international company doing software development at multiple facilities worldwide. They too performed an SEI CMM assessment and made improvements based on this assessment. Schlumberger concentrated on improvements in project management, process definition and control, project planning and control, and training. They began managing their requirements better, resulting in 54% fewer validation cycles (34 versus 15 cycles) before product release. Productivity doubled because there was less rework. On-time delivery of software increased from 51% of the projects to 94% in less than three years. The number of post release defects was reduced from 25% of total defects to 10% in the same time period. Wohlwend observes that 12 to 18 months were required before significant improvements were noticed. He points out that instituting process change is very difficult on existing projects with existing schedules.

Herbsleb (1994) provided statistical results as reported by 13 organizations (both companies and DoD organizations) to show what benefit or value could be gained by organizations involved in CMM-based SPI. The findings by Herbsleb, as shown in Table 3.2, primarily focus on organizations that have improved from CMM Level 1 to Level 2 or Level 2 to Level 3. The costs shown are primarily attributed to the cost of such things as an organization's Software Engineering Process Group (SEPG), the cost of assessments and the cost of training.

Productivity gains were primarily attributed to better requirements elicitation, better software management, and incorporation of a software reuse program. Gains in early detection of defects and reductions in calendar time were primarily attributed to reuse. The number of years organizations had been involved in doing software process improvement ranged from 3.5 years to 6 years. There was no apparent correlation between years of SPI and ROI.

	Number of Organizations	Median	Smallest	Largest
Cost per Software Engineer per Year	5	$1,375	$490	$2,004
Productivity Gains per Year	4	35%	9%	67%
Gains in Early Detection of Defects	3	22%	6%	25%
Reduction in Calendar Time	2	19%	15%	23%
Reduction in Post Release Defects	5	39%	10%	94%
Return on Investment	5	500%	420%	880%

Table 3.2 Improvements from Software Process Improvement (Herbsleb, 1994)

NASA's Software Engineering Laboratory (Basili, 1994 and McGarry 1993), in a 7 year period, has reduced its cost of software development by 55%, decreased its cycle time by 40% and reduced its post release defect rate by 75%. This has been achieved primarily through software reuse (software taken in its entirety). The Software Engineering Laboratory (SEL) takes a different approach than the CMM to improvement; whereas the CMM focuses on improvements in process, the SEL emphasizes improvements in software product based upon the SEL's Experience Factory. However, the SEL recognizes that the CMM is an excellent model of process changes that could be selected to attain product improvement.

The Boeing Space Transportation Systems (STS) Defense and Space Group's process improvement efforts (Yamamura and Wigle, 1997) have been rewarded with a CMM Level 5 rating by the SEI. Boeing's Continuous Quality Improvement (CQI) focused on productivity increase and cycle-time reduction; where some processes reduced cycle time by 50%. Initially 70% of all defects were found during verification & 19% during validation. After peer review inspections were introduced, most defects eliminated before testing. Initially 89% of all defects were found during development, with 11% not found; software processes now find nearly 100% of all defects. Inspections increased the design effort by 25% (4% of total development) which reduced rework during testing by 31%. So a 4% increase in effort returned 31% reduction in rework for 7.75:1 ROI.

Motorola's (Diaz and Sligo, 1997) successes in achieving CMM Level 5 have also been documented. Motorola uses quality, cycle time, and productivity to evaluate their programs because this is what they believe customers value. They use a Six Sigma Quality focus that looks at reject rates as low as a few per million. Their goal is to achieve a 10X reduction in product cycle time to introduce new products quicker. Each level increase of the CMM improves quality by 2X. Higher maturity projects have a better schedule performance index. Defect injection rate is roughly 1/2 for each level of increase; thus, rework for a CMM Level 2 project is 8X that of a level 5 project. Productivity also improves with increasing maturity level, but a noted decrease in productivity between level 2 and 3 appears to be a side effect of asking people to do too many things differently at level 3.

Krasner (1997) cited case studies from several companies who realized benefits from embarking on an SPI program. Motorola India Electronics attained CMM Level 5 in 1993, and realized a 3.5X improvement in productivity in advancing from CMM Level 3 to Level 5, as well as a 40% reduction in cycle time and significant reductions in defects and rework needed.

Millot (1999) reported on ROI as a result of Thomson-CSF as they advanced from CMM Level 1 to CMM Level 2 and, later, to CMM Level 3. The Return-on-Investment (Cost Savings/Cost of Improvement) was a 4.3 ratio measured after the improvements to CMM Level 2, and a 3.75 Ratio measured after the improvements to CMM Level 3. There were also improvements in defect reduction and estimating accuracy. Oldham, et al (1999) measured improvements from progressing from CMM Level 1 to CMM Level 3 for operational flight programs managed at Ogden Air Logistics Center, Utah. The change in cycle time (the time from inception to product release) and cost of development both decreased by 70% after the

improvements; the number of defects decreased from 0.1 Defects/KSLOC measured before the improvements and 0.0 Defects/KSLOC after the improvements; and the ROI (Cost Savings/Cost of Improvement) was 19X measured after the improvements.

In a related study, Walter (1999) presented the results of Oklahoma City Air Logistics Center's achievement of CMM Level 4. Total development time was 13 Months measured before the improvements and 12 Months measured after the improvements (from CMM Level 2 in 1993 to CMM Level 4 in 1996). Number of defects observed per unit output was 3.3 Defects per KSLOC measured before the improvements and 0.3 Defects per KSLOC measured after the improvements (also from CMM Level 2 in 1993 to CMM Level 4 in 1996). The overall cost (investment) of the improvement was 6 Million dollars measured after the improvements (over entire Software Process Improvement from 1989 to 1998), but total savings to the organization was 50.5 Million dollars measured after the improvements, resulting in an ROI of 742 % measured after the improvements.

Porter and DeToma (1999) reported that GTE participated in the 1994 SEI return on investment (ROI) study by providing actual program data. For the five-year period of the study, the results showed that productivity increased 37 percent in terms of source lines of code/hour, error reductions netted 55 percent less defects/thousand source lines of code, and the overall SPI ROI was 6.8. An internal division ROI study conducted in 1995 found similar results with their ROI being 7.8. Other cost reductions have been seen throughout the corporation. The average software defect rate during system integration and test has been significantly reduced over time. Within one division, the level of formal quality assurance support has dropped from being 2.2 percent of the organization (based on head count) to under 1.8 percent (almost a 20 percent reduction). In 1997 that division tailored its software quality assurance (SQA) activities, taking advantage of the maturity of its peer review process, thereby reducing its SQA costs by 50 percent on its programs. In all cases, the improvements in GTE's software process have increased quality while reducing costs, thereby reducing time to market.

Jarzombek (2000) summarized the results of CMM improvements across several Air Force programs. The number of defects that are found after release decreased by 39 to 84 %. The change in cycle time (the time from inception to product release) decreased by 19 to 23 % measured after the improvements. Software development productivity increased by 35 to 75 % measured after the improvements, primarily due to less rework, and software operations and

maintenance costs decreased by 30 to 55%. The Return-on-Investment (Cost Savings/Cost of Improvement) was 400 to 1900 % measured after the improvements.

Pipp (2000) reported on the results of Raytheon Missile Systems advancing from CMM Level 2 to CMM Level 4. They experienced a 144% increase in productivity and a 6:1 ROI resulting from the improvements. They also won a greater percentage of their bids. The company determined that attaining the next CMM Level, Level 5, was a worthwhile goal based on these successes. Harter, Krishnan, and Slaughter (2000) reported on the results from the Navy's SPAWAR Systems Center as they progressed to CMM Level 3. They compared testing for two versions of a product, one of which was done while SPAWAR was still at CMM Level 1 and the other after they attained CMM Level 3. The number of defects found during test phases was 221 measured before the improvements to CMM Level 3, and 64 measured after the improvements. Required effort during the test phase was 12 staff months measured before the improvements and 8 staff months measured after the improvements. The effort required to repair defects discovered during testing was 57.8 staff months measured before the improvements and 17.3 staff months measured after the improvements.

Pitterman (2000) reported on the improvements of Telecordia Corporation as they advanced from CMM Level 3 to CMM Level 5. The number of defects observed per unit output was 120 post-release defects per thousand Function Points measured before the improvements and 3 post-release defects per thousand Function Points measured after the improvements Major software releases that are on time were 91 % measured before the improvements and 99 % measured after the improvements. The resources needed to test a system were 0.62 dollars per SLOC measured before the improvements and 0.50 dollars per SLOC measured after the improvements.

Goyal et al (2001) reported on the positive results of IBM Global Software services in India attaining CMM Level 5. The number of "non-conformances" per project was reduced by about 15%. The reported ROI (cost savings/cost of improvement) was 5.5 from performing the training and other activities needed for improvement.

Pitts (2001) reported on the results of Northrop Grumman, Electronic Sensors and Systems after they attained CMM Level 4. The change in Software Development Productivity was a 40 % increase measured after the improvements. Change in the quantity of defects that are

found after release decreased by 35% after the improvements. Finally, the Return-on-Investment (Cost Savings/Cost of Improvement) was 440 % measured after the improvements.

Bowers (2002) reported on improvements from the F/A 18 Advanced Weapons Laboratory when they reached CMM Level 4. Defect density was very low, 3.8 defects per KSLOC, down from 13.5 before the improvements. Productivity in the design phase was 3.45 man-hours per SLOC, down from 15.7 before the improvements. Design phase cost was $200 per SLOC, down from $725 before the improvements. Life-cycle cost was $400 per SLOC, down from $1,170 before the improvements. The number of test flights was 0.6 flights per KSLOC, down from 3.1 before the improvements.

Diaz and King (2002), in a study they performed at General Dynamics Decision Systems, showed that each higher CMM level of software maturity results in improved software quality and productivity. In progressing from Level 2 to Level 4, defect density, a measure of software quality, was reduced by a factor of almost four, and productivity improved by 90%. Also, ROI improved by 167% from Level 2 to Level 3, and 109% from Level 3 to Level 4. In progressing from Level 4 to Level 5, the differences were not as marked; productivity did improve by about 50% compared to Level 4, but defects were only reduced by 16% and ROI only improved by 14%. This may have been due to the types of projects being studied, however, and progressing from Level 4 to Level 5 does show improvements.

Paulk and Chrissis (2002) published the proceedings of a high maturity workshop with presentations from 27 different organizations on how attaining higher CMM ratings has helped them in the areas of increased productivity, reduced costs, defect reduction, improved estimating accuracy, on-time delivery, and ROI. For example, for Tata Elxsi Ltd., the number of defects observed per unit output was 3 defects / KLOC measured before progressing from CMM Level 4 to Level 5, and 0.75 defects / KLOC measured after Level 5 was achieved. For Lockheed Martin Management Data Systems, software development productivity improved 13 % after progressing from CMM Level 4 to Level 5. For Mastek Ltd., rework costs as a percentage of development costs was 3.9 % measured before progressing from CMM Level 4 to level 5, and 2.1 % measured after achieving CMM Level 5.

In response to a DACS Data Collection Survey (2003), Goldman Sachs reported a 50% productivity improvement, a 90% schedule reduction, and a 1.6 to 1 ROI from using the CMM. HQ USAF reported a defect reduction of 75%, an ROI of 7.5 :1, and a a productivity improvement of

30%, from using the CMM. Urioste (2004) reported on the improvements at Lockheed Martin developing the software for the Tomahawk Missile System. When they advanced from CMM Level 4 to CMM Level 5, software development productivity was 30 % greater, development costs decreased by 20%, and the cost to remove defects decreased by 15% after the improvements.

Table 3.3 summarizes the results of all studies included in the DACS ROI Dashboard© to date for CMM. Although there were a few negative results for productivity, cycle time, and defect reduction, using the CMM gave overall positive results in all areas.

Metric	Data Points	Minimum	Maximum	Median	Mean	Standard Deviation	25th Percentile	75th Percentile
ROI	18	0.14 ratio	19 ratio	6 ratio	5.9 ratio	3.99 ratio	4.3 ratio	6.8 ratio
Productivity	29	5% decrease	350% increase	30% increase	76.83% increase	96.28% increase	15.5% increase	100% increase
Project Cost	2	18% decrease	20% decrease	19% decrease	19% decrease	1.41% decrease	18% decrease	20% decrease
Improvement Cost	2	2% of total effort	3.15% of total effort	2.58% of total effort	2.58% of total effort	0.81% of total effort	2% of total effort	3.15% of total effort
Cycle Time	14	19% increase	90% decrease	38% decrease	36.94% decrease	34.56% decrease	8% decrease	70% decrease
Schedule Variance	10	50% increase	98% decrease	46% decrease	43.3% decrease	40.73% decrease	33% decrease	67% decrease
Quality (% of defects found)	3	90% defects found	100% defects found	94% defects found	94.67% defects found	5.03% defects found	90% defects found	100% defects found
Quality (% defect reduction)	26	-6% defect reduction	100% defect reduction	50% defect reduction	48.62% defect reduction	32.03% defect reduction	26% defect reduction	72% defect reduction
Rework	6	28% decrease	73% decrease	36% decrease	41.5% decrease	16.71% decrease	30% decrease	46% decrease

Table 3.3: Results of CMM Usage from the ROI Dashboard©

The box charts from the ROI Dashboard© in Figures 3.1, 3.2, and 3.3 show the effects on ROI, productivity, and defect reduction respectively. For ROI in Figure 3.1, Ogden Air Logistics Center experienced a 19:1 ROI when they advanced from CMM Level 2 to Level 5 in a 4-year period. All 18 organizations reported a positive ROI, with an average of 5.9:1. The lowest was 1.14:1 from an agency which had advanced from CMM Level 4 to level 5; however, this was due to their reaching Level 5 very quickly after Level 4, and the effective annual ROI would have been much greater. The data suggests that an organization using the CMM as an SPI can expect an ROI between 4:1 and 7:1.

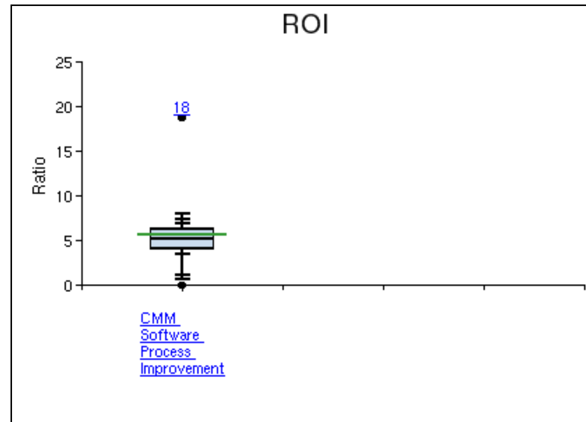

Figure 3.1: ROI Box Chart for CMM

Figure 3.2 shows that one organization, Motorola India, achieved an increase in productivity of 350% in LOC per person-day when they advanced from CMM Level 3 to Level 5 in a period of 3 years. The average was about 75%. The lowest was a near-zero increase from an organization that advanced from CMM Level 3 to Level 4 in a short time. This same organization, however, experienced an overall increase of 2.9:1 from Level 2 to Level 5, and the near-zero result was in part due to the quick transition from Level 3 to Level 4; the Level 4 advances were not realized until they reached level 5. Also, the same organization realized significant reductions in number of defects and rework required when they advanced from CMM Level 3 to Level 4. The data suggests that an organization using CMM as an SPI can expect so see productivity improvements between 15% and 95%.

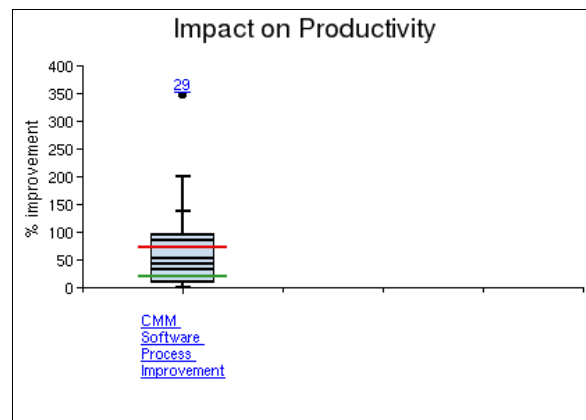

Figure 3.2: Productivity Box Chart for CMM

In Figure 3.3, the best defect reduction result was from Ogden Air Logistics Center, the same organization that achieved the best ROI rating from the CMM. They went from 0.1 defects per KSLOC to 0.0 defects per KSLOC for delivered products. The defect measure used was quality deficiency reports, which are no longer reported. The average improvement was a 48% defect reduction. One company actually had a slight increase in defects when they advanced from CMM Level 3 to CMM Level 5; however, their productivity did improve, and they did experience significant reductions when they advanced from CMM level 5 to CMMI Level 5, the topic of the next Section (Section 3.2.3) of this report. The data suggests that a company using CMM as an SPI can expect between 25% and 70% defect reduction.

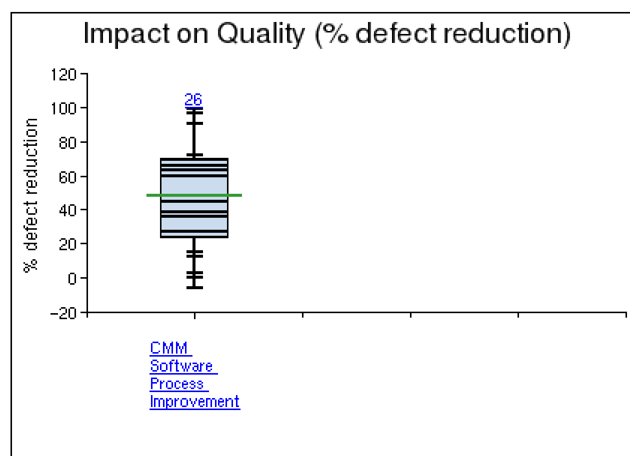

Figure 3.3: Defect Reduction Box Chart for CMM

3.2.3 Capability Maturity Model Integration (CMMI)

According to Wezka, Babel, and Ferguson (2000), success of the CMM spawned a plethora of process maturity models during the 1990s. The CMMI Product Team at the Software Engineering Institute (SEI) along with numerous representatives from Government and industry spearheaded an effort to combine features of the best models, including the CMM, into the CMMI. While the CMM emphasized software engineering, the CMMI also addresses systems engineering, integrated product and process development (IPPD), supplier agreement management, and hardware considerations. There is also a standard assessment method for CMMI, the Standard CMMI Assessment Method for Process Improvement (SCAMPI).

As illustrated in Table 3.4, which is from Version 1.2 of the CMMI released by the CMMI Product Team (2006), there are two representations of the CMMI, continuous and staged. The continuous representation addresses capability levels which are primarily for individual

process areas, while the staged representation addresses overall organizational maturity levels (and is more similar to the CMM). The CMMI is updated periodically, and articles such as the one by Phillips (2007) summarize the changes for each update.

Level	Continuous Representation: Capability Levels	Staged Representation: Maturity Levels
Level 0	Incomplete	N/A
Level 1	Performed	Initial
Level 2	Managed	Managed
Level 3	Defined	Defined
Level 4	Quantitatively Managed	Quantitatively Managed
Level 5	Optimizing	Optimizing

Table 3.4: Comparison of CMMI Levels (CMMI Product Team, 2006)

As is the case for the CMM, many organizations have reported substantial improvements from SPI related to their use of the CMMI. In an SEI technical report, Gibson, Goldenson, and Kost (2006) described in detail the results from more than 30 organizations that have used the CMMI for process improvements. Table 3.5 summarizes the overall median improvements by performance category, and shows substantial improvements, especially for ROI.

Performance Category	Median Improvement
Cost	34%
Schedule	50%
Productivity	61%
Quality	48%
Customer Satisfaction	14%
Return on Investment (ROI)	4:1

Table 3.5: Performance Improvement Over Time By Category (Gibson, Goldenson, and Kosh, 2006)

The SEI Technical Report does summarize results of the contributing companies, but more detailed results for individual companies can be found on the SEI web site for CMMI performance results, http://www.sei.cmu.edu/cmmi/results.html.

More results from individual companies are available from the proceedings of annual CMMI conferences; for example, Freed (2004) described results from Raytheon Network Centric Systems who attained CMMI Level 5 in 2003. They were able to realize a 5% improvement in their overall cost performance index, an 8% improvement in schedule performance index, a 44% decrease in defect density, and a 3:1 ROI. Tower (2004) described marked improvements in the investment bank area from J.P. Morgan from just progressing from CMMI level 1 to CMMI Level 2. Here, an ROI of 5:1 was realized, schedule slippage was reduced by 80%, and defects were reduced by more than 80%. When the firm attained CMMI Level 3, defects were further reduced by 50%. The web site for the proceedings of these conferences is by year; for example, the 2004 conference proceedings are at http://www.dtic.mil/ndia/2004cmmi/.

Goldenson and Gibson (2003) report on several companies who experienced positive results from using the CMMI for process improvement. Accenture stated that an ROI of 5:1 could be realized for some of their programs. Boeing (Australia) realized a 33% decrease in costs of correcting defects, and Lockheed Martin (M&DS) realized a 15% decrease in costs of correcting defects, as well as a 4.5% decline in overhead rates and a 20% reduction in unit software costs.

There are several reports that describe the results of using the CMMI in single companies or organizations. Sheldon (2003) reported on the results of using the CMMI for process improvement at several Raytheon sites. A site which reached CMMI Level 3 realized an ROI of 6:1 and a 100% increase in early defect containment rates. Another site that reached Level 3 reduced rework costs by 42%. Henry, et al (2003) described the results of Motorola GSG in India from attaining CMMI Level 5. The cost of rework was 6.4 % of Effort measured after the improvements, which exceeded the goal of 8%. The change in cycle time (the time from inception to product release) was 1.62 ratio of baseline cycle time to CMMI Level 5 cycle time measured after the improvements, which exceeded the goal of 1.6 times. The change in software development productivity was 1.17X improvement measured after the improvements which did not meet goal of 1.5X, but did show improvement.

Scott (2004) described reductions in cycle times as NCR Self Service in Scotland progressed through CMM and, later, CMMI levels. Among other results, cycle time was reduced from 80 weeks to about 26 weeks when the company reached CMM level 3. Singh (2004) describes a company that attained CMM Level 3, and was in the process of attaining CMMI Level 2. Defects found during the acceptance test phase was about 44 % measured after the

improvements compared to before the improvements. (Causal Analysis and Resolution (CAR) identified process steps injecting defects.)

Weszka (2004) presented the results of using the CMMI at three Lockheed Martin organizations. At Lockheed Martin Systems Integration in Owego, NY, software development productivity improved 126 % in transitioning from CMM Level 5 to CMMI Level 5. The number of defects delivered to the customer was reduced 80 % after transitioning from CMM Level 5 to CMMI Level 5. At Lockheed Martin Maritime Systems and Sensors, Radar Division, the number of defects delivered to the customer was also reduced by 80% after transitioning from CMM Level 5 to CMMI Level 5. At Lockheed Martin Maritime Systems and Sensors, Undersea Division, Total major defects observed was 0.15 major defects / KSLOC measured before the improvements and 0.051 major defects / KSLOC measured after transitioning from CMM Level 5 to CMMI Level 5. Richter (2004) measured the cost per function as DG Systems advanced from "below CMM Level 2" to "CMM Level2" to "CMMI Level 3 almost satisfied". The cost per function was only 80% of the cost of below CMM Level 2 when CMM Level 2 was attained; however, it was reduced to 48% of the "below CMM Level 2" cost when the company reached "almost CMMI Level 3". Finally, Vu (2005) assessed the improvements for Boeing as they progressed from CMM Level 5 to CMMI Level 5. There was about a 35% decrease in schedule required for similar programs, as well as a 4% decrease in maintenance costs and a 10% increase in software reuse.

According to Reifer (2007), special considerations must be given to those organizations that have attained CMMI Level 5. Here, SPI activities can not be justified based on improvements that occur at higher levels; cost avoidance and productivity tend to remain "flat". At Level 5, organizations focus on optimizing existing processes instead of developing new ones. However, these organizations can make a case for SPI activities if they are aligned with improving business and emphasizing finding and correcting defects early in a program.

Table 3.6 summarizes the results of all studies included in the DACS ROI Dashboard© to date for CMMI as an SPI. Like the CMM, using the CMMI gave overall positive results in all areas.

Metric	Data Points	Minimum	Maximum	Median	Mean	Standard Deviation	25th Percentile	75th Percentile
ROI	9	2 ratio	13.3 ratio	3 ratio	4.64 ratio	3.55 ratio	2.25 ratio	5.5 ratio
Productivity	12	5% increase	250% increase	39% increase	57% increase	67.5% increase	13.5% increase	66.5% increase
Project Cost	2	20% decrease	40% decrease	30% decrease	30% decrease	14.14% decrease	N/A	N/A
Improvement Cost	1	1.1% of total effort	1.1% of total effort	1.1% of total effort	1.1% of total effort	N/A	N/A	N/A
Cycle Time	5	15% decrease	50% decrease	38% decrease	32.6% decrease	14.62% decrease	17.5% decrease	45% decrease
Schedule Variance	3	35% decrease	50% decrease	40% decrease	41.67% decrease	7.64% decrease	35% decrease	50% decrease
Rework	1	60% decrease	60% decrease	60% decrease	60% decrease	N/A	N/A	N/A
Quality (% of defects found)	1	98% defects found	98% defects found	98% defects found	98% defects found	N/A	N/A	N/A
Quality (% defect reduction)	20	0.5% defect reduction	95% defect reduction	48.5% defect reduction	47.64% defect reduction	29.21% defect reduction	25.5% defect reduction	67% defect reduction

Table 3.6: Results of CMMI Usage from the ROI Dashboard©

Figure 3.4 shows the ROI results for both CMM and CMMI. The average ratio of 4.64:1 for CMMI is less than that of the CMM; however, several organizations had already attained high CMM levels and were measuring changes from transitioning from the CMM to the CMMI, sometimes at the same maturity level. For example, one company attained a 2:1 ROI when they moved from CMM Level 5 to CMMI Level 5. The data suggests that an organization using the CMMI as an SPI can expect an ROI between 2:1 and 6:1.

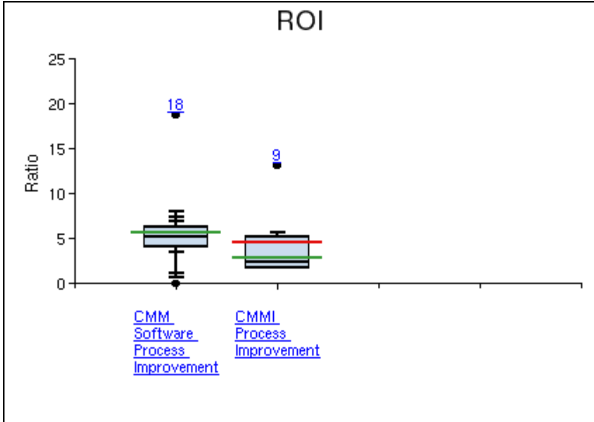

Figure 3.4: ROI Box Chart for CMM and CMMI

Figure 3.5 shows the productivity results for both CMM and CMMI. The average increase of 57% for CMMI is, as in ROI, less than that of the CMM; however, as in the case of ROI, several organizations had already attained high CMM levels and were measuring changes

from transitioning from the CMM to the CMMI, sometimes at the same maturity level. For example, two companies incurred productivity improvements of only 17% and 25% as they moved from CMM Level 5 to CMMI Level 5. The data suggests that an organization using the CMMI as an SPI can expect a productivity increase between 20% and 70%.

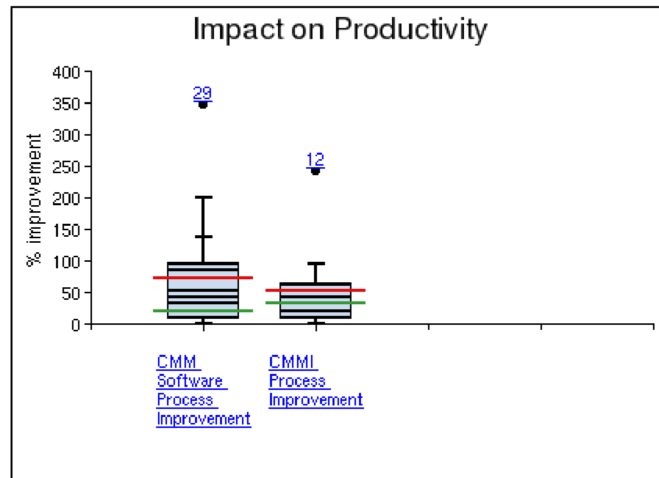

Figure 3.5: Productivity Box Chart for CMM and CMMI.

Figure 3.6 shows the defect reduction results from both the CMM and CMMI. It is very interesting to note the similarity of the statistical averages and medians between the CMM and CMMI. The lower values for CMMI are for companies that moved to a CMMI level from an equivalent CMM level (and, in one case, from a higher CMM level). The data suggests that an organization using the CMMI as an SPI can expect a defect reduction between 25% and 65%.

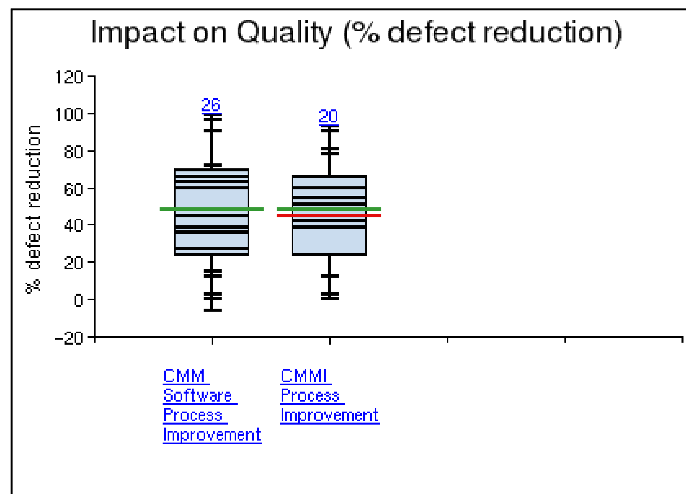

Figure 3.6: Quality Box Chart for CMM and CMMI

3.2.4 Personal / Team Software process (PSP/TSP)

The PSP and TSP were developed in the late 1990s by Humphrey (2000) and other SEI associates to apply CMM principles to individual and team development respectively. The idea behind the PSP and TSP is that well-qualified people and properly formed teams of qualified people can develop quality products, which should improve ROI. Indeed, Webb and Humphrey (1999) report significant improvements after implementing TSP for a program at Hill AFB, Utah. Defects removed prior to integration testing improved to 21 defects per thousand source lines of code (KSLOC) from 13 before TSP was implemented. Productivity during testing improved to 0.22 test days per KSLOC from 0.94 to 2.89 test days per KSLOC before TSP was implemented. Defects found during the acceptance test phase was 0.07 to 1.89 Defects/KSLOC measured before the TSP improvements and 0.0 Defects/KSLOC measured after the improvements

Ferguson (1999) reported that AIS Corporation realized improvements from using the PSP. When the PSP was used, the number of defects decreased by more than 50%, and productivity increased by about 20%, primarily because less testing time and less rework was needed. In addition to reducing cost; hence, improving ROI, the resultant products were of higher quality. An added benefit was more accuracy in cost and schedule estimating. In another study, Ferguson (2000) reported that Advanced Information Systems, Inc (AIS) observed that software productivity increased from 8.1 LOC per hour to 9.5, and defects per KLOC were reduced from 1.0 defects per KLOC to 0.42 after the PSP was employed. McAndrews (2000) reported on results by Teradyne, Boeing, and TIS Corporation in using the TSP. In all three cases, the actual number of post-released defects markedly decreased for TSP projects. Also, for both Teradyne and TIS, productivity improved by about 16% and costs were reduced by approximately 15% which shows positive ROIs.

Munson (2002) showed that use of the TSP in one organization reduced total costs by 60%. Using TSP, the organization expended more effort in the beginning of the program, during requirements and design, but spent less effort during testing, and much less on post-test activities which involved fixing defects. Training costs of implementing TSP were offset by savings after only 1600 lines of code are produced. Also, by eliminating most of the time needed to correct defects in released products, an organization can concentrate instead on exploiting new business opportunities.

Davis and Mullaney (2003) reported the results of 13 organizations that used the TSP for software development. Overall, these organizations averaged a 6% schedule overrun for their programs. This compares with an industry average of 100% using traditional methods, and a large number of cancellations resulting from excessive schedule overruns. The quality of the software developed by TSP organizations is remarkable. The TSP organizations delivered software with 0.06 defects per thousand lines of code compared to CMM Level 1 organizations using traditional methods that averaged 7.5 defects per thousand lines of code. Even CMM Level 5 organizations could only average 1.05 defects per thousand lines of code before they employed TSP. Hoffman (2003) reported on results from Northrop-Grumman using the PSP, along with CMMI Level 5, to improve performance on a program they were managing. The number of defects observed per unit output was 6.6 per KSLOC measured before the improvements and 2.1 per KSLOC measured after the improvements. The ROI (cost savings/cost of improvement) was a 13.3 ratio measured after the improvements based on time saved on defect resolution. They also reported an increase in customer satisfaction which is "priceless".

Kimberland (2004) reported dramatic results from using the TSP on a Microsoft project. There was a 35% project cost savings, and unit test defects were reduced from 25 per KLOC to 7 per KLOC. The team also had a high score on Microsoft's internal project quality index. Humphrey (2004) studied the effects of using TSP for CMM Level 5 companies on defect reduction. Defects per thousand lines of code (KLOC) were reduced from 1.05 to 0.06 when the TSP was used.

A criticism of quality improvement is a perception that productivity will be reduced because of the purported extra effort needed to insure high quality software. However, using TSP, organizations improved their productivity by 78%. The use of TSP, therefore, is a "win-win" practice since quality is enhanced while cost is reduced.

Table 3.7 summarizes the results of all studies included in the DACS ROI Dashboard© to date for PSP/TSP usage as an SPI. Like the CMM and CMMI, using the PSP/TSP gave overall positive results in all areas.

Metric	Data Points	Minimum	Maximum	Median	Mean	Standard Deviation	25th Percentile	75th Percentile
ROI	2	2.6 ratio	13.3 ratio	7.95 ratio	7.95 ratio	7.57 ratio	N/A	N/A
Productivity	1	16% increase	16% increase	16% increase	16% increase	N/A	N/A	N/A
Project Cost	1	35% decrease	35% decrease	35% decrease	35% decrease	N/A	N/A	N/A
Schedule Variance	2	82% decrease	100% decrease	91% decrease	91% decrease	12.73% decrease	N/A	N/A
Quality (% defect reduction)	5	52% defect reduction	100% defect reduction	85% defect reduction	78.8% defect reductio n	18.89% defect reduction	60% defect reduction	94.5% defect reduction
Quality (% of defects found)	2	96.8% defects found	99.6% defects found	98.2% defects found	98.2% defects found	1.98% defects found	N/A	N/A

Table 3.7: Results of PSP/TSP Usage from the ROI Dashboard©

Figure 3.7 shows the defect reduction results from using the PSP/TSP. Although there are limited reported data on results from PSP/TSP, the average defect reduction of 80% seems impressive. The majority of organizations using the PSP/TSP were also advancing in CMM or CMMI levels and attributed the defect reductions to CMM (or CMMI) and PSP/TSP usage combined, so the actual impact of PSP/TSP alone is probably less than Figure 3.7 would indicate. Still, based on results form organizations using only the PSP/TSP as an SPI, using the PSP/TSP are likely to result in a reduction in defects.

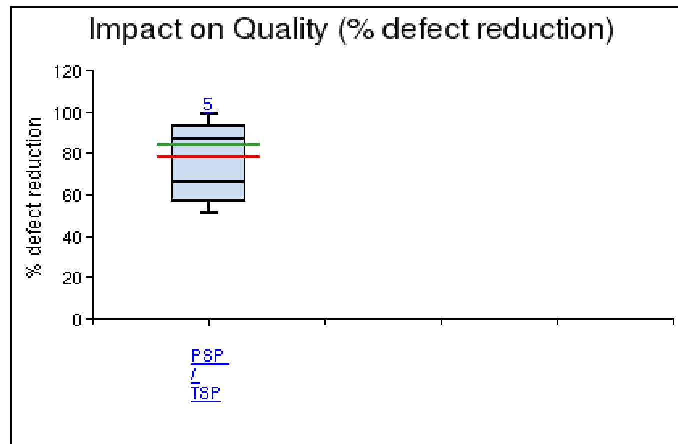

Figure 3.7: Quality Box Chart for PSP/TSP

3.3 Results of Specific Process

While Section 3.2 addressed general practices and benefits, this Section addresses benefits from five specific practices that have been shown to improve ROI for organizations. The five specific practices are inspections, software reuse, Cleanroom software development, agile methods, and systems engineering.

3.3.1 Inspections

Inspections are formal processes that are intended to find defects in software and other products at or near the point of insertion of the defect, thereby using fewer resources for rework. This is achieved by inspecting the output product(s) of each operation in the development process to verify that it satisfies the exit criteria of the operation. Defects are defined as any deviations from the exit criteria of any operation. Inspections can be performed on any product (e.g., test plans, procedures, users manuals, code, code fixes) to improve defect detection efficiency of any process that creates the product. Inspections are based on the inspection process developed by Fagan (1986) and are typically referred to as Fagan inspections. Inspections are a more rigorous and formal process than walk-throughs; they are checklist-oriented in most instances.

Users of the method reported significant improvements in quality, development costs and maintenance costs. Jones (1996) estimates that the average software company in the United States releases products with 15% of the defects still in the product. Companies that rigorously employ inspections are approaching 1% defects remaining in a released product. Jones observed that with 15% defects in released products, you can never have satisfied customers.

Doolan (1992) described the Fagan inspection process. An inspection is organized by a moderator who is appointed by the Software Quality Assurance organization. The moderator reviews the inspection article and checks that it satisfies the entry criteria of the operation. The moderator assembles an inspection panel of no more than 5 people (including the creator). The inspectors attend a 20-30 minute kick-off meeting to discuss the objective of the inspection and to distribute inspection material. Roles are assigned to some of the inspectors. Inspectors then study the material and note any defects. Checklists are used to stimulate defect finding. A meeting of less than 2 hours is held in which every defect is logged including defects discovered at the logging meeting. A causal analysis meeting is then organized to discuss some of the errors

uncovered in the inspection, and to propose possible ways to prevent this type of error in the future.

Fagan (1986) reported that developers who participate in the inspection of their own product actually create fewer defects in future work. Since inspections formalize the development process, productivity and quality enhancing tools can be adopted more easily and rapidly. Between 1976 and 1984, on their System 370 software, IBM was able to double the lines of code shipped and reduce by two-thirds the number of defects per KSLOC (thousands lines of code) by utilizing inspections. Fagan stated that design and code inspection costs amount to 15% of project costs. On four projects reported by Fagan, defects discovered after release ranged from 0.0 defects per KSLOC to 0.3 defects per KSLOC. One project reported a 25% reduction in development costs; another reported a 9% reduction in inspection costs versus walk-throughs; and a third project reported a 95% reduction in corrective maintenance costs. In 1979, Fagan reported that 12% of programmer resources were for fixing post-shipment bugs. Fagan estimated that the cost of finding and fixing defects early is 10 to 100 times less than it would be during testing and maintenance.

Doolan (1992) utilized Fagan's inspection technique at Shell Research's Seismic Software Support Center (SSSC) for validating and verifying requirements. Doolan observed that previously 50% of the 300 to 400 faults found each year in SSSC's software were enhancement requests that would have been dealt with had more care been taken in specification and requirements analysis of the product. Given an average cost of approximately $3,500 to fix an error (for manpower, computer time, testing time, configuration management, fault-report database updates, user notification and generation of release notifications), Doolan assessed the cost of this non-conformance at 3.5 man years. He measured the payback in terms of costs saved for every hour invested in inspections. Doolan estimated that fixing defects in released software costs as much as 80 times more than during the specification stage. He estimated that for each hour of inspection, 30 hours are recouped, for a yearly average saving of 16.5 staff-months. Non-quantifiable benefits were that inspections promote teamwork and team spirit, and prevent the introduction of the same errors in the future.

At the Jet Propulsion Laboratory (Kelly, 1992), a modified version of Fagan inspections was used for software requirements (R1), architectural design (I0), detailed design (I1), source code (I2), test plans (IT1) and test procedures (IT2). They modified the inspection process, adding a third hour to the inspection logging session to discuss problem solution and resolve

open issues raised in the inspections. A 1.5 day course in formal inspections was the only initial cost to implement inspections. Experience from the 203 inspections measured (23 R1, 15 I0, 92 I1, 34 I2, 16 IT1 and 23 IT2) showed a higher density of defects in early life cycle products than in later ones. Kelly was able to approximate defects found per page as y = $3.19e^{-0.61x}$, where x= 1, 2, 3, 4 for R1, I0, I1 and I2 respectively. The average cost to fix defects close to their origin found during inspections was 0.5 hours versus 5 to 17 hours required to fix defects found during formal tests, because defects found during formal tests require detection and tracing of the defect and all associated documents, and then retesting. The cost in staff hours to find a defect during inspections was 1.1 hours and 0.5 hours to fix the defect.

Madachy (1995) described utilization of inspections at Litton Data Systems. Litton has experienced a 30% reduction in the number of errors found during systems integration and test. 400 people at Litton have been trained and 600 inspections have been performed. On one project they have experienced a 50% reduction in integration effort. Madachy estimated that 2.3 staff hours are saved in systems testing for every inspection hour. 73% of the 600 inspections have produced a positive return and inspections account for 3% of the total project effort.

Cardiac Pacemakers Incorporated (Olson, 1995) utilized inspections to improve the quality of its life critical software. Olson estimates that if the cost to fix a defect during design is 1X, then fixing design defects during test is 10X and in post-release is 100X. He estimates the effort to fix and verify a defect once detected in the process is 0.25 to 0.5 hours during requirements and design, using the inspection process, versus 5-10 hours during systems integration and test, utilizing testing, which is a 10-20:1 ratio. Costs incurred were the cost of inspections (5-15% of the total project), startup costs and overhead (e.g., SEPGs). Cost reductions are the estimated cost by phase without inspections minus the actual costs by phase with inspections. The estimated ROI is 7:1 with $600-$700k in savings. Olson observed that inspections remove 70-90% of faults; the rest are removed with tests.

Bull HN Information Systems (Weller, 1993) conducted 6,000 inspections on its GCOS 8 system. They estimated that code inspections before unit test found 80% of the defects and inspections after system test found 70% of the defects. They have concluded that inspections can replace unit testing, but not later stages of testing. Rooijmans (1996) reported that Philips TV observed the following changes as a result of using Fagan inspections: The number of defects observed per unit output was 5.41 Test Defects per Kilobyte measured before the inspection improvements and 2 Test Defects per Kilobyte measured after the improvements. The overall

cost investment of the improvement was 0.8 staff years measured after the inspection improvements, but there was a total savings of 6 staff years as a result of using inspections, which results in an ROI of 650%. Lee (1997) documented the lessons learned from application of formal inspections on Lockheed Martin's space shuttle onboard software project. On their projects, they have been able to achieve error detection rates of 85 % to 90%.

O'Neill (2003) compared the results of using practices of ad hoc programming (AHP), structured software engineering (SSE), and disciplined software engineering (DSE), which includes use of inspections, on development defect detection and leakage, and test defect detection and leakage. AHP corresponds to CMM Level 1, SSE to CMM Levels 2 and 3, and DSE to CMM Levels 4 and 5. The results of a National Software Quality Experiment conducted during the late 1990s showed that DSE results in the most defects being detected during development (requirements, design, and coding), and fewer defects after testing. While the ROI dollars are not always evident from DSE, the much higher quality of the software is evident, so organizations using DSE with its accompanying emphasis on inspections are expected to have greater customer satisfaction.

El Emam (2005) showed that inspections are effective in that the effort needed to correct defects using design or code inspections is only 25% of the effort needed to correct defects during testing. The ROI from using inspections instead of testing alone, in terms of percentage savings, is 11% for pre-release costs, 26% for post-release costs, and 38% for customers. The use of inspections results in a 3% schedule savings, which shows that quality improvements from inspections do not add time to a software project. Freimut (2005) described the results of using inspections at Siemens AG, Germany. The cost of rework was a 67 % mean decrease measured after the improvements for inspections performed in the analysis phase, the cost of rework was a 47% mean decrease measured after the improvements for inspections performed in the design phase, and the cost of rework was a 12% mean decrease measured after the improvements for inspections performed in the development, or coding phase.

Table 3.8 summarizes the results of all studies included in the DACS ROI Dashboard© to date for using inspections as an SPI.. Use of inspections gave overall positive results in all areas.

Metric	Data Points	Minimum	Maximum	Median	Mean	Standard Deviation	25th Percentile	75th Percentile
ROI	15	2 ratio	30 ratio	4.8 ratio	6.84 ratio	6.62 ratio	4.23 ratio	7 ratio
Productivity	2	37% increase	350% increase	193.5% increase	193.5% increase	221.32% increase	N/A	N/A
Project Cost	1	8% decrease	8% decrease	8% decrease	8% decrease	N/A	N/A	N/A
Improvement Cost	1	3% of total effort	3% of total effort	3% of total effort	3% of total effort	N/A	N/A	N/A
Quality (% defect reduction)	5	63% defect reduction	100% defect reduction	99% defect reduction	86.6% defect reductio n	18.12% defect reduction	67% defect reduction	100% defect reduction
Quality (% of defects found)	6	50% defects found	100% defects found	86% defects found	77.5% defects found	22.07% defects found	50% defects found	93% defects found
Rework	4	12% decrease	67% decrease	39% decrease	39.25% decrease	23.39% decrease	21.5% decrease	57% decrease

Table 3.8: Results of Using Inspections from the ROI Dashboard©

Figure 3.8 shows the ROI results from using Inspections. The median ratio of 4.80 is probably more representative of actual results than the mean of 6.84 since one organization's 30:1 ratio skews the mean significantly. The median value is also consistent with an extensive study by O'Neill (2003), the National Software Quality Experiment, which involved many Government, Commercial, and DoD contractor organizations. The ROIs for these organizations averaged 4.69, 4.65, and 3.69 respectively. The data suggests that organizations can expect a 3.5:1 to 5:1 ROI when inspections are used as an SPI.

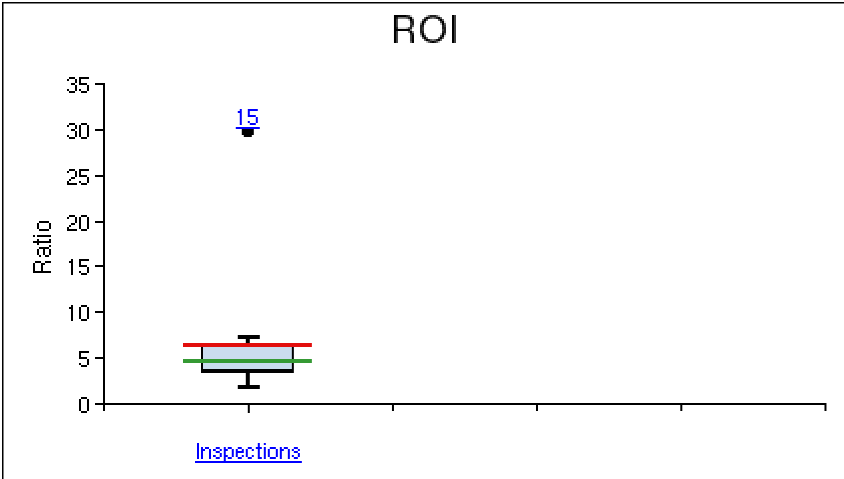

Figure 3.8: ROI Box Chart for Inspections

Figure 3.9 shows the quality results from using Inspections in terms of percentages of defect reduction and defects found. The median percentages of 96% for defect reduction and 84% for percentage of defects found indicate that use of inspections as an SPI can dramatically reduce defects. The available data suggests that organizations can expect to reduce defects by 90 to 99% using inspections, and can increase the percentage of defects found by 80% to 90%.

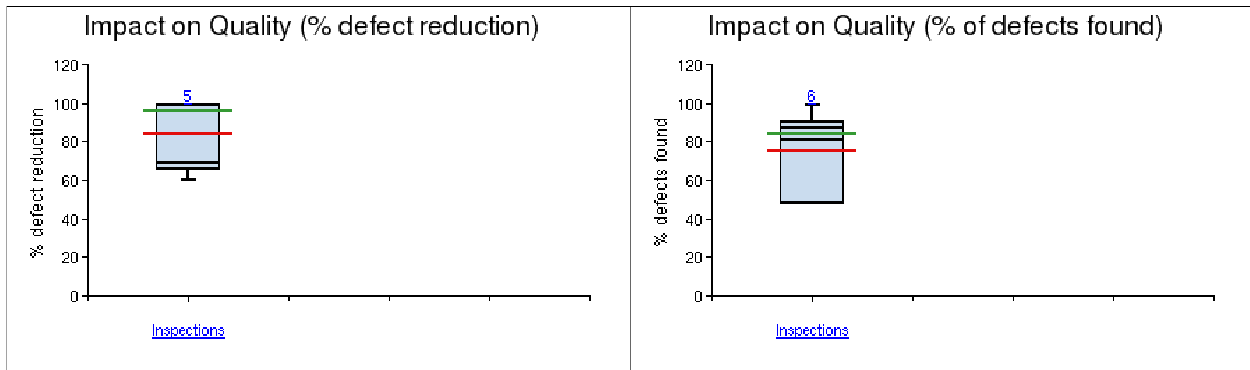

Figure 3.9: Quality Box Charts for Inspections

3.3.2 Software Reuse

As described by Lim (1994), work products are the products or by-products of the software development process. Work products include designs, specifications, and code and test plans. Reuse is the use of existing work products elsewhere within a project or on other projects. Since software development schedules, estimates and costs are heavily influenced by the amount of new code that has to be designed and developed, and if software development is on the critical path of a project, reusing work products can have a significant positive impact on costs and schedules for a project.

A significant portion of the literature concentrates on systematic reuse in which organizations design software to be reusable. For example, at two of the divisions of Hewlett Packard (HP) (Lim, 1994), reuse was a critical ingredient in achieving productivity and quality objectives. Lim noted that because work products are used several times, the accumulated fixes in each use results in a higher quality product. One division experienced a 51% reduction in defects (from 4.1 to 2.0 defects/KSLOC) in all code and a 57% increase in productivity (from 700 to 1,100 SLOC/staff month) on 68% reused code. The other division experienced a 24% defect reduction (from 1.7 to 1.3 defects/KSLOC), a 40% increase in productivity (from 500 to

700 SLOC/staff month) and a 42% reduction in cycle time (from 36 to 21 calendar months) on 31% reused code. Reused code had 0.4 defects per KSLOC. The author noted that the additional costs to create reusable work products ranged from 111% to 480% of a work product that did not consider reuse. By phase, the percent increase in effort was 22% for investigation, 20% for design, 17% for code, 5% for testing and 5% for repairing. Lim reported that with reuse others have stated that high level design costs increase by 10%, detailed design by 60% and code and unit test by 25%. However, the relative cost of reuse ranged from 10% to 63% of the costs of a build from scratch project. One division of HP has been involved in systematic reuse for 10 years. Gross costs for that period were $1 million with a savings of $4.1 million for a ROI of 410%. The break-even point on the start up costs ($300,000) was in the second year. Another division has been involved in reuse for 8 years. Gross costs were $2.6 million and savings were $5.6 million for a ROI of 216% with a break-even in the sixth year.

O'Connor (1994) described Rockwell International's reuse experience within command and control systems. They used the Synthesis Methodology, as developed by the Software Productivity Consortium (SPC). Reuse is integral to this methodology, which forms the foundation for a product family and associated production processes. It defines a systematic approach to identifying the commonalities and variability necessary to characterize a standardized product line as a domain. A domain is a product family and process for producing instances of that family. Commonalities reflect work that can be accomplished once and reused to create any product. Variability specifies compile time parameters to customize the application. O'Connor stated that the cost of creating a reuse domain is approximately equivalent to the cost of making a hand-crafted system in that domain of the same size. Although no specific savings were noted, the author did state that the benefits of this reuse methodology were improved productivity, improved product quality, improved responsiveness to customer needs, lower bids on projects, and institutionalization of shared knowledge and expertise among systems in a business area.

In a seven year period the NASA Software Engineering Laboratory, as reported by McGarry (1993) and Basili (1994), increased the average level of reuse by 300% from about 20% to nearly 79%. At the same time, the error rate has decreased by 75% from 4.5 to 1 error/KSLOC, and the cost of software per mission has decreased by 55% from about 490 staff months to about 210 staff months.

Joos (1994) described the reuse efforts on two pilot projects at Motorola. In one pilot project, to encourage reuse, a cash reward incentive program was established. Each time an asset is added to the reuse library, a $100 reward is paid to the developer. Each time a reuse asset is consumed, an award proportional to the savings is given to the developer and the person or organization reusing the software. On a second project involving compiler development and a compiler tool test suite, an 85% reuse rate was achieved with a 10:1 productivity improvement.

Leach (1997) reported on the results of reuse for 312 projects in the aerospace industry prior to 1993. For projects employing reuse, software development productivity increased by 20%, the number of customer complaints decreased by 20%, the mean time to repair (correct defects) decreased by 25%, and development time decreased by 25%, on the average. Leach also wrote that a 1993 study of Japanese companies reported similar results from reuse, including a 20% reduction in training time needed.

Poulin (1997) reported on the benefits of reuse for several companies. Raytheon reported a 50% increase in productivity from 60% reuse in COBOL programs. DEC reported that cycle times were reduced by factors ranging from 3 to 5 based on 50% to 80% reuse. Toshiba reported a 20 to 30% reduction in defects based on 60% reuse, as Matsumura (1991) had noted previously. A study of nine other companies showed reuse resulted in an 84% decrease in cost, a 70% reduction in cycle time, and reduced defects.

Ezran, Morisio, and Tully (2002) explained reuse as a systematic software development practice. It not only includes "building blocks", or components, but also requirements and architecture similarities. They gave several examples of companies that have benefited from reuse not only in reducing costs and schedules (hence, improving ROI), but also in improving software quality. For example, Hewlett Packard reduced defects by 74% while improving productivity by 57% for a project, while NEC improved productivity by 600% and quality by 300%.

Clements and Northrop (2002) explored reuse as a feature of software product lines, which are "sets of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way". Reuse not only includes software components, but other "strategic assets" including requirements, architecture, performance models and analyses, business cases, software tools, test cases, plans, and data, and personnel skills and training. Northrop has been doing ongoing research at the SEI on product lines, and in a later

presentation, Northrop (2007) gave several instances of benefits from reuse in product lines. For example, Cummins reduced product cycle effort from 250 person-months to "a few person months", and a Raytheon organization reduced development time and cost by 50% while increasing productivity by a factor of 7 and quality by a factor of 10. Time to field was 9 years measured before the improvements and 3 years measured after the improvements. Development staff size was 210 developers measured before the improvements and 30 developers measured after the improvements.

L. Jones (1999) also reported positive results from using product lines. According to Jones, the Swedish naval defense contractor, CelsiusTech, reported a reversal in the hardware-to-software cost ratio from 35:65 to 60:20 that now favors the software. Hewlett Packard has collected substantial metrics showing two to seven times cycle time improvements with product lines. Motorola has realized a four times cycle time improvement with 80 percent reuse.

With the obvious benefits of software reuse, as noted by Card (1994), why have so many reuse programs failed? He has observed that reuse programs have achieved 30 to 80% reuse; however, others have failed. He has concluded that the economics are such that the amount of reuse depends on how well the reuse products match the needs of the reuse consumer, the skill and knowledge of the consumer about reuse, and the degree of similarity between producer's and consumer's requirements. From a cultural viewpoint, hindrances include the "not invented here" syndrome and resistance to change. Overcoming these cultural hindrances requires training, incentives, good project management, and good reuse measurement.

Lougee (2005) concluded that reuse can save time and cost, improve ROI, and enhance safety at the same time; however, reuse will not be effective unless a rigorous process is used. The elements of this process include identifying the purpose and goals, cataloging and analyzing existing or proposed artifacts, comparing alternative approaches and selecting the best approach, planning thoroughly, modifying strategy when appropriate, and mitigating risks. Lougee also emphasized the need to involve stakeholders throughout the entire reuse process.

Table 3.9 summarizes the results of all studies included in the DACS ROI Dashboard© to date for using reuse (including product lines) as an SPI. Employing reuse gave overall positive results in all areas.

The majority of reported results of reuse initiatives focus on the impacts on productivity and cycle time. Figure 3.10 shows the productivity improvement results from employing reuse.

As in Figure 3.8 (ROI for Inspections), the median value of 69.5% increase is probably more representative of actual results than the mean of 279.92% since one organization's 1400% skews the mean significantly. Also, as Table 3.8 shows, there was significant variance in results. The data suggests that organizations can expect a 30% to 80% increase in productivity depending on the degree and experience with reuse.

Metric	Data Points	Minimum	Maximum	Median	Mean	Standard Deviation	25th Percentile	75th Percentile
ROI	3	4 ratio	10 ratio	4.1 ratio	6.03 ratio	3.44 ratio	4 ratio	10 ratio
Productivity	12	20% increase	1400% increase	69.5% increase	279.92% increase	416.74% increase	45% increase	400% increase
Project Cost	2	84% decrease	84% decrease	84% decrease	84% decrease	0% decrease	N/A	N/A
Improvement Cost	1	111% of total effort	111% of total effort	111% of total effort	111% of total effort	N/A	N/A	N/A
Cycle Time	13	25% decrease	98.4% decrease	66.7% decrease	58.55% decrease	22.07% decrease	40% decrease	75% decrease
Quality (% defect reduction)	1	25% defect reduction	25% defect reduction	25% defect reduction	25% defect reductio n	N/A	N/A	N/A
Quality (% of defects found)	1	90% defects found	90% defects found	90% defects found	90% defects found	N/A	N/A	N/A

Table 3.9: Results of Reuse from the ROI Dashboard©

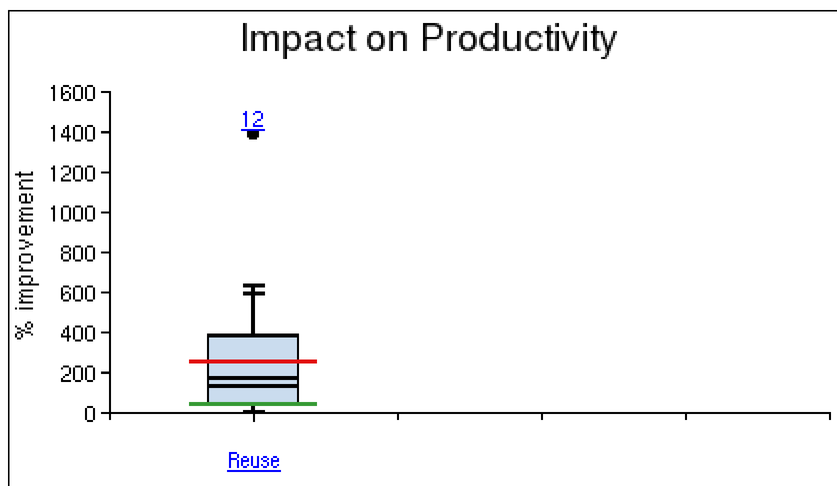

Figure 3.10: Productivity Box Chart for Reuse

Figure 3.11 shows the cycle time reduction results from employing reuse. Here, the mean and median values are relatively close, a mean decrease of 58.6% and a median decrease of

66.7%. The results with higher values are from reuse associated with product lines and from more recent studies. The data suggests that reuse, especially across product lines, can reduce development times by 50 to more than 100%.

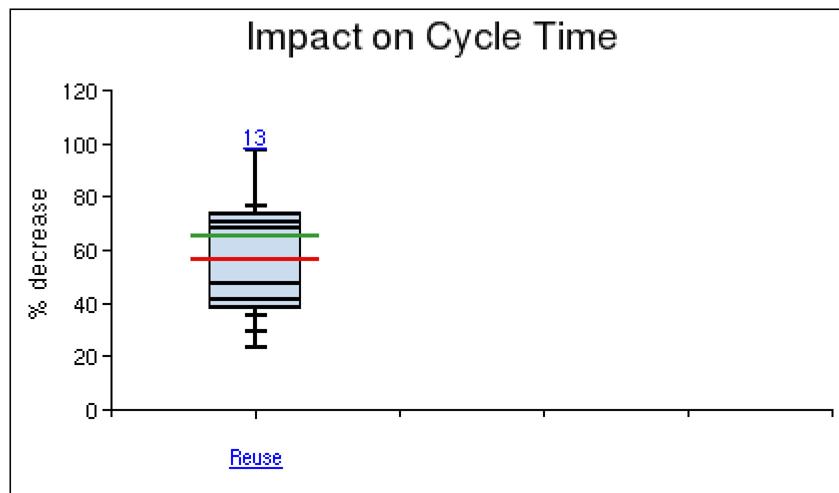

Figure 3.11: Productivity Box Chart for Cycle Time

3.3.3 Cleanroom Software Development

The objective of the Cleanroom methodology is to achieve or approach zero defects with certified reliability. As described by Hausler (1994), the Cleanroom methodology provides a complete discipline within which software personnel can plan, specify, design, verify, code, test and certify software. In a Cleanroom development, correctness verification replaces unit testing and debugging. After coding is complete, the software immediately enters system test with no debugging. All test errors are accounted for from the first execution of the program with no private testing allowed. As opposed to many development processes, the role of system testing is not to test in quality; instead, the role of system testing is to certify the quality of the software with respect to the systems specification. This process is built upon an incremental development approach; Increment I_{n+1} elaborates on the top down design of increment I_n. The Cleanroom process is built upon function theory where programs are treated as rules for mathematical functions subject to stepwise refinement and verification.

Cleanroom specifications and designs are built upon box structure specifications and design. Box structure specifications begin with a black-box specification in which the expected behavior of the system is specified in terms of the system stimuli, responses and transition rules.

Black boxes are then translated into state-boxes which define encapsulated state data required to satisfy black box behavior. Clear box designs are finally developed which define the procedural design of services on state data to satisfy black box behavior. Team reviews are performed to verify the correctness of every condition in the specification. During the specification stage, an expected usage profile is also developed, which assigns probabilities or frequency of expected use of the system components. During system correctness testing, the system is randomly tested based on the expected usage of the system. In this process, software typically enters system test with near zero defects.

The Cleanroom process places greater emphasis on design and verification rather than testing. In this process, as in inspections (see Section 3.3.1), errors are detected early in the life cycle, closer to the point of insertion of the error. In some of the earliest reported findings on Cleanroom development, Mills (1987) observed that, with a Cleanroom methodology, 90% of the defects were found before the first execution of the code versus 60% with traditional developments. 2.65 errors per KSLOC were observed on a 53 KSLOC program with an observed productivity of 400 SLOC per month. He observed that errors from this process are much easier to fix, stating that they take 20% the time to fix as compared to traditional software.

Linger (1993) showed that on 15 projects with a combined total of 5 million lines of code, the average error rate of 3.3 errors per KSLOC has been observed from first execution versus 30 to 50 errors per KSLOC on traditional projects. Linger also observed that the errors in Cleanroom verification are typically simple mistakes, not design mistakes. Drastically reduced maintenance costs result from Cleanroom development. He claimed that experienced Cleanroom teams with subject matter experts involved can achieve a substantially reduced product development lifecycle.

Hausler (1994) claimed that the time spent in specification and design is greater than in traditional projects, but the time spent in testing is less. Overall, the lifecycle cost is much lower than industry averages, and Cleanroom project schedules are less than or equal to traditional schedules. Hausler claimed that productivity improvements of 1.5 to 5.0 have been observed over traditional projects. In 17 projects performed at IBM using the Cleanroom process, the code developed exhibited a weighted average of 2.3 errors per KSLOC through all testing measured from the first execution. This can be compared against 25 to 35 errors per KSLOC in traditional software.

Basili (1994) stated that at the NASA SEL, the time to understand the Cleanroom methodology was approximately 26 months. This time was from first training to the start of the second Cleanroom project. He observed error rates of 4.3 to 6 errors per KSLOC versus 7 errors per KSLOC on traditional projects.

At the US Army's Life Cycle Software Engineering Center at Picatinny Arsenal, Sherer (1996) reported a productivity increase from 121 SLOC per staff month to 559 SLOC per staff month (a 362% increase) using Cleanroom software engineering over the life cycle of their project. Cumulative failures over the same period were 1.14 per KSLOC. Sherer estimated an ROI of 20.8:1 from introduction of the Cleanroom methodology, where the costs of the methodology were training and coaching[2] costs. Training and coaching costs added 17.3% labor to the project costs. Sherer attributed significant improvements in job satisfaction, team spirit and team morale to the methodology.

Pressman (2005) said that Cleanroom software development has not gained widespread acceptance. Indeed, few articles have appeared in software books and journals since the mid 1990s. However, Pressman stated that the potential benefits of Cleanroom software development far outweigh the investment required to implement Cleanroom software development, even in organizations where there is cultural resistance to using it. It results in extremely low failure rates that would be difficult, if not impossible, to achieve using less formal methods.

Table 3.10 summarizes the results of all studies included in the DACS ROI Dashboard© to date for using Cleanroom development as an SPI. Based on limited data, employing Cleanroom development gave positive results in all areas. The available data suggests that use of the Cleanroom methodology is likely to be quite beneficial.

Metric	**Data Points**	**Minimum**	**Maximum**	**Median**	**Mean**	**Standard Deviation**	**25th Percentile**	**75th Percentile**
ROI	1	20.8 Ratio	20.8 Ratio	20.8 Ratio	20.8 Ratio	N/A	N/A	N/A
Productivity	2	15% Increase	360% Increase	187.5% Increase	187.5% Increase	243.95% Increase	N/A	N/A
Cycle Time	1	74% Decrease	74% Decrease	74% Decrease	74% Decrease	N/A	N/A	N/A
Quality (% of defects found)	1	90% Defects Found	90% Defects Found	90% Defects Found	90% Defects Found	N/A	N/A	N/A

Table 3.10: Results of Cleanroom Development from the ROI Dashboard©

[2] Coaches are recognized experts who work after the training to keep the entire team on a common level of expertise.

3.3.4 Agile Development

Agile software development is a conceptual framework for undertaking software engineering projects. There are a number of different methods for agile development, most of which emphasize minimizing risk by developing software in short time boxes, or iterations, which typically last one to four weeks. The methods usually emphasize face to face communication over written documents and working software as a measure of progress. Stapleton (1997) studied the effects of using DSDM, an agile method, for the user interface for a newspaper tracking system program consisting of about 880 Function Points. Sysdeco (of England), the developing organization, found that effort productivity changed from 5.3 hours per Function Point measured before the improvements to 1.5 hours per Function Point measured after the improvements using this method.

Probably the most publicized agile development method is eXtreme Programming (XP). XP assumes that change is a "way of life" and emphasizes adaptability instead of predictability. The fundamental XP values are communication, simplicity, feedback, and courage. It involves performing work in small teams with a customer continuously present on site. The cycle time is only about three weeks between releases. XP practices include stories, pair programming, simple design, "test first", and continuous integration. The use of agile development methods, including XP, is a process improvement method that may result in improved productivity and quality and, hence, a positive ROI. Drobka, Noftz, and Raghu (2004) showed that Motorola was able to improve productivity using XP compared to incremental and waterfall development methods. While use of incremental development improved productivity by 162% over the waterfall method, the four XP projects improved productivity by 265%, 283%, 352%, and 385% over the waterfall method. Test coverage also was very good, but quality varied. For two of the projects, one was better and the other worse than the quality of traditional methods. Since the four projects were pilot projects; however, quality may improve with time.

Manzo (2002) described successes from using Code Science, a tool based on agile methods including XP, on a large program called Odyssey. For one project, XP and the waterfall approach were used side-by-side for the same project. At one time, the XP project was fully completed while the waterfall project was not even 50% completed. On several projects, including Odyssey, dramatic reductions in cost and defects were realized when Code Science was

used. Productivity rates averaged 35 lines of code per hour, which is much greater than that of traditional methods.

Hodgetts and Phillips (2003) presented the results of adopting XP at an internet business startup. They compared the results from two similar efforts for which XP was not utilized and for which XP was (later) utilized. The delivery time was 20 months measured before the XP improvements and 12 Months measured after the improvements. Total defects observed were 508 defects measured before the improvements and 152 defects measured after the improvements. Effort was 207 developer-months measured before the improvements and 40 developer-months measured after the improvements.

Layman (2004) studied the effects of adopting XP at Sabre Airline Solutions. The same team performed two releases of a product, one before using XP and one after using XP. For XP, defects found during test phases was 1.0 relative defects per thousand lines of executable code (KLEC) measured before the improvements and 0.35 relative defects per KLEC measured after the improvements. The number of defects delivered to the customer was 1.0 relative defects per KLEC measured before the XP improvements and 0.64 relative defects per KLEC measured after the improvements (by four months after release). Software development productivity was 1.0 relative KLEC per person month measured before the XP improvements and 1.46 relative KLEC per person month measured after the improvements. Maurer and Martel (2002) studied a web-based application software development effort that was performed by a group of nine programmers using XP. They compared the number of new lines of code, number of methods, and number of classes per person-month using XP and previous efforts not using XP. They noted an increase of productivity of 66% for lines of code, 302% for number of methods, and 283% for number of classes. The writers showed how the XP practices improved productivity, but stated there was little hard data available at the time to show if XP improved ROI for other programs.

Pair programming, a common XP practice, has been studied by several researchers. Nosek (1998) conducted a study at Temple University using 15 full time, experienced programmers working for a maximum of 45 minutes on a challenging problem. Five worked individually and 10 worked in pairs. Temple University observed that the time from project inception to release was 42.6 minutes measured before the improvements and 30.2 minutes measured after the improvements. (The total effort of pairs was 65% more than the solo programmers.) Pair programming was also studied by Cockburn (1999), who conducted a

controlled experiment at the University of Utah that investigated the economics of pair programming. When pair programming (versus individual programming) was used, total development time increased by 15%; however, there was a 15% decrease in total defects measured after the improvements (comparing pair programmers to individual programmers). Also, there was a 20% reduction in lines of code required for the program written. A lesson learned was that the increased cost due to pair programming was offset by reduced defects in the testing and support phases measured after the improvements.

Williams (2000) also studied the impact of using pair programming on software development for students at the University of Utah. It was found that, once they adjusted to the practice, pair programming teams developed programs in about 60% of the time it took individual programmers, and developed programs that only had half as many errors. Furthermore, at least 90% of the programmers enjoyed pair programming more than individual programming, except in rare cases working with particular incompatible partners.

Nawrocki and Wojciechowski (2001) described an experiment conducted at Poznan University of Technology comparing pair programming using XP with two variants of individual programming; one variant was based on Humphrey's Personal Software Process (PSP) and the other on individual programming using XP. Test subjects were students who wrote four programs using C, C++, or Pascal. Six programmers followed PSP; 5 programmers followed XP without pair programming; and 5 pairs followed XP with pair programming. The results of the experiments were that total development time was 3.13 mean hours measured before the pair programming improvements and 2.44 mean hours measured after the improvements. Defects found during the acceptance test phase averaged 2.53 before the pair programming improvements and 2.1 after the improvements. However, software development productivity decreased from 57 mean LOC per person-hour measured before the improvements to 48.25 mean LOC per person-hour measured after the pair programming improvements.

Ciolkowski and Schlemmer (2002) reported the results of a case study conducted at the University of Kaiserslautern where students were asked to change a quiz program written in Java. Six student teams participated; three teams used pair programming and the other three worked individually. Effort was 24.5 mean hours measured before the pair programming improvements and 27 mean hours measured after the improvements. Source lines of code (SLOC) were 4040 SLOC measured before the improvements and 3890 SLOC measured after the improvements for

the same functionality. Neither of the results was statistically significant, but they show the same general trends as other studies done before 2002.

Rostaher and Hericko (2002) studied the results of using pair programming for a software development organization in Slovenia. The organization, called the FJA OdaTeam, noted that the time from project inception to release was 6.05 hours measured before the improvements of pair programming and 6.00 hours measured after the improvements; however, effort was 6.05 person hours measured before the improvements and 12.00 person hours measured after the improvements. Baheti, Gehringer, and Stotts (2002) performed a controlled experiment at the University of North Carolina collecting data from a class project with teams of 2 to 4 students. There were 9 collocated teams without pairs and 16 collocated teams with pairs, and 8 distributed teams without pairs and 5 distributed teams with pairs. Software development productivity was 15.1 LOC per hour measured before the pair programming improvements and 14.8 LOC per hour measured after the improvements for collocated teams, but this was not statistically significant. For distributed teams, software development productivity was 21.1 LOC per hour measured before the improvements and 18.5 LOC per hour measured after the improvements, but this was also not statistically significant.

Arisholm, et al (2007) reported the results of an experiment conducted at Simula Research Laboratory on pair programming for simple and complex Java maintenance tasks. Statistically significant results are summarized here: For the simple task, duration was 88 adjusted mean minutes measured before pair programming improvements and 70 adjusted mean minutes measured after the improvements, and effort was 88 person-minutes measured before the improvements and 140 person-minutes measured after the improvements. For the complex task, effort was 61 person-minutes measured before the improvements and 129 person-minutes measured after the pair programming improvements, but programmers or teams completing all tasks correctly improved from 55 percent measured before the improvements to 81 percent measured after the improvements.

Lierni (2007) described a survey performed for over 700 organizations to determine the current state of agile software development. Over 80% of those organizations do use agile development methods, and more than 25% of those organizations showed significant improvements from using agile methods in four areas; they averaged a 60% reduction in

schedule, a 26% reduction in cost, a 55% reduction in defects, and a 55% improvement in productivity.

Scrum (a term derived from the game of Rugby) is another popular agile development method. Stutzke (2005) described Scrum as a method that uses a form of time boxing which divides software development into 30 day periods called sprints. Schatz and Abdelshafi (2005) realized quality improvements using Scrum for projects in their company, Primavera Systems. The company realized a 30% improvement in software quality from using Scrum on a company project. Also, the project was delivered four months earlier than the planned release date.

While several studies have shown that agile methods can improve ROI, these methods should not be looked at as a panacea for software development. Boehm and Turner (2004) show that there are advantages and disadvantages to agile methods and disciplined methods, and show how to determine the optimal mix of both methods for a specific type of program. In few if any cases are this mix purely agile methods or purely disciplined methods.

Table 3.11 summarizes the results of all studies included in the DACS ROI Dashboard© to date for using agile development and its related methods, such as XP and pair programming, as an SPI. Although there were some negative results, especially in productivity, employing agile development averaged positive results in all areas.

Metric	**Data Points**	**Minimum**	**Maximum**	**Median**	**Mean**	**Standard Deviation**	**25th Percentile**	**75th Percentile**
ROI	1	5 Ratio	5 Ratio	5 Ratio	5 Ratio	N/A	N/A	N/A
Productivity	13	53% Decrease	418% Increase	2.4% Decrease	63.84% Increase	136.95% Increase	26% Decrease	112.5% Increase
Cycle Time	6	6% Increase	45% Decrease	21% Decrease	18.5% Decrease	18.62% Decrease	1% Decrease	29% Decrease
Quality (% of defects found)	1	17% Defects Found	17% Defects Found	17% Defects Found	17% Defects Found	N/A	N/A	N/A
Quality (% defect reduction)	6	-2% Defect Reduction	99.9% Defect Reduction	22.5% Defect Reduction	31.32% Defect Reduction	36.34% Defect Reduction	9% Defect Reduction	36% Defect Reduction

Table 3.11: Results of Agile Development from the ROI Dashboard©

Figure 3.12 shows the impact on productivity from using agile development. The median of near zero shows that as many agile development efforts had negative results as positive results. Most of the negative results were for pair programming, an XP activity. Pair programming usually resulted in decreased cycle times and improved quality, but the results varied. When other agile practices were used in addition to pair programming (or without pair programming),

the results for productivity were positive. The most positive results occurred for those organizations that were experienced in agile methods, had management support, and worked on programs suitable for the use of agile methods. The data suggests that organizations can expect productivity improvements when agile methods are used as an SPI when they are appropriate. A reference like Boehm and Turner (1984) can help an organization to determine what types and to what degree agile methods should be used for specific programs or projects.

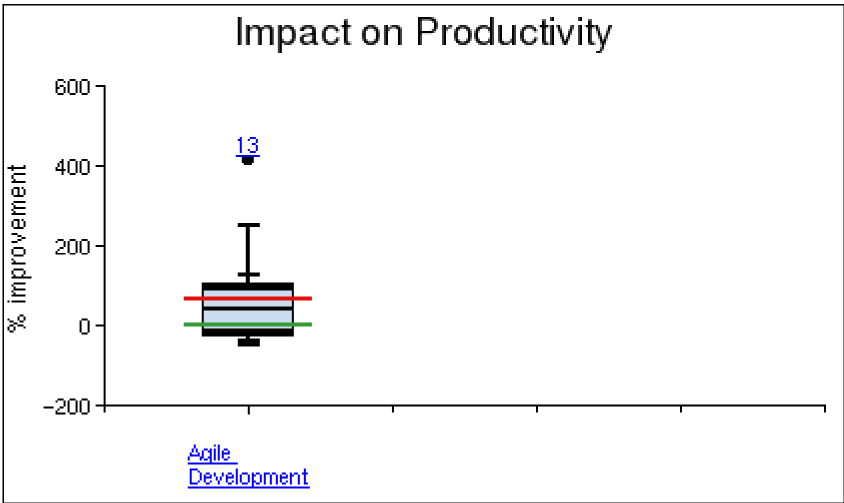

Figure 3.12: Productivity Box Chart for Agile Development

3.3.5 Systems Engineering

As defined in CMMI Version 1.2 (CMMI, 2006), systems engineering is "the interdisciplinary approach governing the total technical and managerial effort required to transform a set of customer needs, expectations, and constraints into a product solution and to support that solution throughout the product's life". This includes the definition of technical performance measures, the integration of engineering specialties toward the establishment of product architecture, and the definition of supporting lifecycle processes that balance cost, performance, and schedule objectives.

Kossiakoff and Sweet (2003) devote a chapter of their book to software systems engineering, the application of systems engineering principles and methods to software system design and engineering. They state that the continuing demand for complex software-dominated systems may accelerate efforts to introduce systems engineering methods into software development. CMMI Version 1.2 (CMMI, 2006) defines software engineering itself as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and

maintenance of software". By whatever term it is known, application of systems engineering to software efforts can be seen as an SPI area.

Verma (2007) described the results of applying systems engineering and architecture techniques to software development projects at IBM. Of the 62 projects, 27 used the techniques and 35 did not. Software development productivity was 15 Function Points (FPs) per thousand project-hours measured before the systems engineering and architecture improvements and 27 FPs per thousand project-hours measured after the improvements. The number of defect hours after delivery to the customer was 52 defect hours during warranty measured before the improvements and 25 defect hours during warranty measured after the improvements. Cycle time in days per function point averaged about 1.1 before the improvements and about 0.4 after the improvements. This study provides strong evidence that application of systems engineering to software projects improves ROI.

Table 3.12 summarizes the results of the study included in the DACS ROI Dashboard© to date for using systems engineering as an SPI. Based on limited data, employing systems engineering gave overall positive results. As is the case for Cleanroom development, the available data suggests that systems engineering is a useful SPI for software efforts, and more data should be available as time progresses.

Metric	**Data Points**	**Minimum**	**Maximum**	**Median**	**Mean**	**Standard Deviation**	**25th Percentile**	**75th Percentile**
Productivity	1	80% Increase	80% Increase	80% Increase	80% Increase	N/A	N/A	N/A
Quality (% of defects found)	1	52% Defects Found	52% Defects Found	52% Defects Found	52% Defects Found	N/A	N/A	N/A

Table 3.12: Results of Systems Engineering from the ROI Dashboard©

3.4 Secondary Benefits of Improvement Efforts

This Section presents literature that discusses secondary benefits of software improvement efforts. Many such benefits have been noted in the literature. For example, Strassman (1990) believed that proper utilization of IT will result in gains in market share, better prices, reduced inventories, and highly motivated employees. According to Humphrey (1991), lower software professional turnover, improved employee morale, improved company image,

improved customer satisfaction, improved quality of work life, and improved schedule performance resulted from process improvements at Hughes Aircraft. Lipke and Butler (1992) discuss process improvement efforts at the Oklahoma City Air Logistics Center (OC-ALC) and describe as intangible benefits of their efforts increased communications, increased customer satisfaction, and on-time software delivery.

Dion (1993) stated that second order effects of their process improvement efforts are improved competitive position, higher employee morale, lower absenteeism, and lower attrition. These efforts also have less late and over budget projects. Significant benefits of software reuse noted by Lim (1994) are that experienced people can concentrate on developing products that less experienced people can reuse and shortened time-to-market. Brodman and Johnson (1995) discuss "spillover" benefits from process improvement, including improved morale and confidence of developers, less overtime, less employee turnover, and improved competitive advantage. The authors describe how increased productivity can mean a more competitive edge in bidding on contracts, and can increase company's capacity to do work and thus perform more work within a given period of time. Meeting schedule and cost projections can translate to customer satisfaction, repeat business, and decreased time to market, and improved product quality translates to more dollars on the bottom line.

Olsen (1995) discussed the "Quality Chain Reaction" as being that improved quality results in reduced cost (less rework), fewer mistakes, fewer delays, better designs, more efficient use of resources and materials, improved productivity, and larger market share with better quality and lower price. Curtis (1995) suggested a number of secondary factors to consider when evaluating process improvement, since SPI reduces development costs, reduces rework costs, and improved estimates. These factors include savings from less terminated projects, less missed delivery dates, and less employee turnover. Missed delivery dates results in penalties, lost market share, lost revenue, overruns, and less repeat customer business. The author also notes that experience has shown that SPI results in fewer crises, less overtime, more business, and higher project bonuses.

McGibbon (1997) examined the benefits of formal methods and documented how, like Cleanroom software engineering, formal methods exhibit less rework after products are released than in traditional software. According to Diaz and Sligo (1997), the most significant cost benefit from Motorola's improvement efforts occurred when projects finished early, allowing the company to apply more resources to obtaining more business. Motorola believes that

productivity is directly related to their ability to win new programs in their DoD business, and drives their profitability in emerging commercial products.

Porter and DeToma (1999) reported that, in addition to improved ROI from using SPI, GTE experienced some qualitative differences including employee retention and hiring. GTE found that it is more attractive to work in a more mature organization which has employee involvement in improvement efforts. The SPI work was becoming "real" to the people.

Jensen (2003) discussed the results from an experiment in pair programming, one of the major practices associated with XP and agile development. According to Jensen, managers sometimes believe that having two programmers do the work of one is wasteful, and that programmers prefer to work in isolation. Jensen found, however, that programmers enjoyed working in pairs and that they can effectively and efficiently produce a quality product. Their error rate was only a third of that of programmers working alone.

Burke and Howard (2005) discussed the FAA's process improvement initiatives and the results on the people and the organization. The FAA started SPI as an organizational mandate, but this had limitations in that it did not address individual project needs. Later, SPI efforts were combined with knowledge management, which promoted free sharing of information among units within the FAA organization. The combination has resulted in individual groups being dedicated to improving their processes, and in organizational barriers being broken down; people are now cooperating instead of competing.

Baddoo, Hall, and Jagielska (2006) examined motivational and success factors for software developers in a CMMI Level 5 organization. These developers rate user satisfaction as the primary measure of a software product after one year, well ahead of cost and schedule concerns. Also, developers in a high maturity organization tend to be more highly motivated, which, in turn, results in more successful software development efforts.

Some organizations have attempted to quantify some of the secondary benefits. Yamamura and Wigle (1997) stated that Boeing's improvement efforts results in excellent performance, high customer satisfaction, satisfied employees, and a 97% to 100% award fee for 6 years. The authors state that employee satisfaction grew from 74% to 96% because of the improvements. They assert that employees take pride in their accomplishments as they dramatically reduce defects. Goyal (2001) reported an increase in developer and customer satisfaction resulting from reaching CMM Level 5. The index for satisfaction of users of Key

Process Area of project planning was 65 % measured before the improvements and 90 % measured after the improvements. User satisfaction with tracking and oversight increased from 64% to 80%.

Whether these secondary benefits can be measured directly or not, they do supply additional benefits from using SPI that will provide positive returns to an organization. They can and should be considered in planning for and justifying SPI efforts.

3.5 Risks From Software Improvement

Some may believe that SPI is a magic potion for successful software development. However, there is some literature that addresses various factors and risks that need to be considered when implementing various software management practices.

Utilization (reuse) of commercial off the shelf (COTS) software is a very popular topic today and, as noted by Carney and Oberndorf (1997), is a major emphasis of DoD acquisition organizations. The authors recognize that use of commercial products is one of the remedies that might enable us to acquire needed capabilities in a cost effective manner. The authors believe that using COTS components may be beneficial or cause more problems. The authors discussed ten issues that should be considered before selecting COTS components:

a. Recognize that COTS components are one potential strategy in a complex solution space. Trade offs need to be made when selecting COTS products.

b. The term COTS should only be applied when source code is unavailable and maintenance is impossible except by its vendor. COTS should not be confused with other terms such as Government off the Shelf (GOTS) and Modified off the Shelf (MOTS).

c. A COTS bias will have an impact on specification of requirements. Someone must choose which requirements can bend to the exigencies of the marketplace and which cannot.

d. Understand the COTS impact on the integration process. The phrase "plug and play" is the unspoken motivator for much of the current interest in COTS. However, vendors tend to keep data details private.

e. Understand the COTS impact on the testing process. What types of testing at the unit level and system level are possible utilizing COTS? What types of testing at the unit level and system level are necessary?

f. Realize that a COTS approach makes a system dependent on the COTS vendors.

g. Realize that maintenance is not free. Version upgrades must be supported - ignoring new releases can not survive in the long run.

h. COTS have been designed to work stand alone, not integrated. A COTS based system is still a system with its own requirements - the system needs to be designed, integrated, tested and maintained.

i. Recognize hidden costs: understanding COTS products as system components; market research to find COTS products; product analyses to select among alternatives; licenses and warranties; product integration; revisions; coordination of support vendors; recovery when a vendor discontinues a product or goes out of business.

j. The shift from building from scratch to integration of ready-made components is a significant paradigm shift for programmers and system developers. It is not just a technical change; there are organizational impacts; it is a shift from a producer to consumer mentality.

Dorofee (1997) discussed the lessons learned by the SEI in application of its risk management program. The SEI's risk management program identifies five functions in risk management: identify, plan, track, control, and communicate. Lessons learned include: (1) Put risks in writing since written risks are harder to ignore than verbal concerns., and use a risk information sheet to document risks; (2) Perform quantitative analysis only when necessary; it is only needed for risks that require numerical justification or rationale for mitigation planning; (3) Group related risks; (4) Prioritize and sort risks since not all risks can be mitigated; (5) Have metrics and measures to track both the risk and the mitigation plan. Spreadsheets that summarize all open risks are good for an overall view of the program's risks; and (6) Use databases to document risks, problems, schedules and change tracking, collect and analyze lessons learned.

Carney, Morris, and Place (2003) reported that some programs like the Hubble Space Telescope program successfully employed more than 30 COTS programs. However, there are other programs where COTS contributed to a lack of success. The SEI has developed a COTS Usage Risk Evaluation (CURE) tool to evaluate the risk of using COTS with large software programs. Some examples of risk factors that should be considered are successful use on other programs, vendor support (present and future), and existence of user groups.

Boehm (2002) explained that there are risks involved in using agile methods, especially without any planning or discipline. Both agile and plan-driven methods have a home ground for which they work best. For most programs, there is a balance somewhere between the two

methods, and agile development does present risks; it is not a nostrum for all software development difficulties. (Of course, that is true of any SPI method.)

4. Detailed Research

As Section 3 demonstrates, there is a significant amount of evidence to show that, with a properly run SPI program, software development organizations can dramatically reduce cycle time, reduce development costs, improve quality, reduce maintenance costs, improve employee morale, and improve company competitiveness. Rozum (1993) developed a benefit index for quantitatively measuring the benefits of an SPI program.

This Section presents a spreadsheet model of the savings and return on investment (ROI) that can be achieved with the SPI programs outlined in Section 3. This spreadsheet enhances a previously developed COCOMO size and cost estimating spreadsheet. With this spreadsheet, one can evaluate several different process improvement methods, evaluate primary and secondary benefits, as well as estimate the size and cost of the software. The spreadsheet was developed under Microsoft Excel®. (Details on using this spreadsheet can be found in Appendix A.)

Table 4.1 identifies many of the key parameters necessary to model ROI estimates for the SPI methods in this Section. The values of these parameters represent values before improvements are made. Many of the SPI methods require training and incur other costs that are a function of the number of personnel in the development staff. "Project Staff Size" identifies the staff size being modeled. The value shown (28 personnel) is derived directly from the schedules in the COCOMO cost estimation spreadsheet. "Lines of Code" are calculated directly from the COCOMO size estimation spreadsheet. Since many process improvements increase the productivity of development staff, code size is an important driver in the ROI model. "Average Staff Hour Cost" is derived directly from the COCOMO cost estimation spreadsheet and is used to convert labor manpower estimates to costs. Many SPI methods reduce the number of defects induced into a product; thus reducing rework costs. Defect rates are measured in terms of "Average Defects per KSLOC." Several sources such as (Basili, 1994) and (Curtis, 1995) have shown that 7 defects per KSLOC is a typical defect rate throughout the development process for new code. The "Software Defect Removal Efficiency" measures the percentage of the induced defects that are removed during the development process. Jones (1996) states that the average United States software development organization removes only 85% of defects induced prior to release to the customer. That means that of the 7 defects/KSLOC induced, 1.05 defects/KSLOC

are left in the product when the customer receives it. SPI methods improve this efficiency and thus reduce maintenance costs.

SPI Model Parameter	Model/Typical Value
Project Staff Size	28 Personnel
Lines of Code	39,967 LOC
Average Staff Hour Cost	$39.00
Average Defects per KSLOC in New Code	7
Software Defect Removal Efficiency	85%

Table 4.1: Parameters to Software Process Improvement Model

4.1 Modeling the Cost Benefit of Software Process Improvement

In Section 3.2, SPI was shown to provide a positive ROI for most software development organizations. Jones (2000) has provided the most comprehensive model for establishing ROI from SPI. Jones has defined the sequential stages organizations typically move through on their way to maturity. In that model, Jones has identified for a given staff size what the costs are per employee to achieve each stage, the average length of time organizations remain at each stage, the defect improvement realized, the productivity gains achieved and the schedule improvements that can be achieved by each stage.

Using the parameters of Table 4.1, Table 4.2 shows the ROI parameters defined by Jones. The first column shows each stage of the Jones model. The "Pre-SPI" row has been added to show the starting values for a given organization with the parameters in Table 4.1. For each stage, Table 4.2 shows the estimated cost to achieve that stage, the number of calendar months required to achieve that level, the reduction in the estimated number of defects in the product as a level is attained, the gains that can be achieved in productivity at each successive level, cycle time improvements resulting from the SPI, the reductions in development and maintenance costs, and the ROI potential of the improvements.

The values computed in Table 4.2 are derived from table values of improvement percentages as described by Jones. In this simple example alone, the project could have been produced with 95% fewer defects, at greater than 3X the productivity, in 70% less time, 70% less

cost and with 95% less maintenance costs had the product team been utilizing the best software process

Stage	Estimated Cost To Reach Stage	No. of Months To Reach Stage	Estimated Number of Defect	Productivity LOC/Day	Schedule Length	Project Development Costs (1)	Project Maintenance Costs	ROI (2)
Pre-SPI			279.77	6 LOC/Day	27 Calendar Months	$2,886,543	$475,681	
Stage 0 Assessment/ Baseline	$3,359	2 Months	279.77	6 LOC/Day	27 Calendar Months	$2,886,543	$475,681	
Stage 1 Management	$50,392	3 Months	251.79	6 LOC/Day	27 Calendar Months	$2,886,543	$428,113	0.89:1
Stage 2 Methods/Practices	$50,392	4 Months	125.90	8 LOC/Day	24 Calendar Months	$2,248,220	$214,057	8.64:1
Stage 3 New Tools	$167,974	4 Months	113.31	10 LOC/Day	22 Calendar Months	$1,606,442	$192,651	5.74:1
Stage 4 Infrastructure	$33,595	3 Months	107.64	11 LOC/Day	21 Calendar Months	$1,443,794	$183,018	5.68:1
Stage 5 Reusability	$16,797	4 Months	16.15	18 LOC/Day	17 Calendar Months	$823,992	$27,453	7.79:1
Stage 6 Industry Leadership	$50,392	6 Months	15.34	19 LOC/Day	17 Calendar Months	$780,174	$26,080	6.85:1
Total Impact	$372,902	26 Months	95% Reduction	222% Increase	37% Reduction	73% Reduction	95% Reduction	

Table 4.2: ROI Benefits of Software Process Improvement

Demonstrating the value of the software CMM, Jones (2007) has provided a number of "rules of thumb" for estimating various attributes of projects, such as effort estimation, schedule estimation, defect injection rates, and defect removal efficiencies based on the CMM level of the organization performing the project. Jones utilizes function points extensively as the measure of size in his rules of thumb calculations. These rules of thumb have been implemented in the spreadsheet and are shown in Table 4.3.

The top half of Table 4.3 demonstrates the overall rules of thumb parameters as set forth by Jones. Reading from left to right, Jones shows for each CMM level the estimated per person cost to reach that CMM level, the number of calendar months to achieve that CMM level, the project schedule (computed as the number of Function Points to the "Exponent"), the defect injection rate per function point, the percentage of defects that will be removed, and productivity (function points per staff month).

From the parameters of Table 4.1, the "Lines of Code" size is converted to function points (assuming 128 LOC/FP for the C language), for an estimate of 312 function points. The bottom half of Table 4.3 then apply the rules of thumb for the sample project. As can be seen, Jones would estimate that achieving CMM Level 5 from Level 1 will result in a 28% decrease in

the project schedule, a 140% reduction in defects being induced into the software, an 11X reduction in defects to be found and repaired in maintenance, and a 2X reduction in effort to complete the project.

	Cost/person and Time to CMM Level		Project Schedule	Defect Potential and Removal Efficiency		Monthly Productivity
	Estimated Cost	Calendar Months		Defect Potential	Removal	FP Per
CMM Level	To Reach Level	to Reach CMM Level	Exponent	Per FP	Efficiency	Staff month
CMM Level 1	$0.00	0 Months	0.45	6.00	85%	3.00
CMM Level 2	$3,500.00	18 Months	0.44	4.50	89%	3.50
CMM Level 3	$3,000.00	18 Months	0.43	4.00	93%	5.00
CMM Level 4	$3,500.00	12 Months	0.42	3.00	95%	7.50
CMM Level 5	$2,000.00	12 Months	0.41	2.50	97%	9.00
This Project	Estimated Cost	Calendar Months			Defects Remaining	Est. Staff
	To Reach Level	to Reach CMM Level	Est. Schedule Length	Est. Defects	At Product Release	Months Required
CMM Level 1	$0.00	0 Months	13 Calendar Months	1873 Defects	281 Defects	104 Staff Months
CMM Level 2	$124,245.53	18 Months	13 Calendar Months	1405 Defects	155 Defects	89 Staff Months
CMM Level 3	$106,496.17	18 Months	12 Calendar Months	1249 Defects	87 Defects	62 Staff Months
CMM Level 4	$124,245.53	12 Months	11 Calendar Months	937 Defects	47 Defects	42 Staff Months
CMM Level 5	$70,997.44	12 Months	11 Calendar Months	781 Defects	23 Defects	35 Staff Months
Total Impact	*$425,984.67*	*60 Months*	*26% Improvement*	*140% Improvement*	*1100% Improvement*	*200% Improvement*

Table 4.3: ROI Benefits of Software CMM

4.2 Modeling the Benefits of Specific processes

This Section shows how the quantitative benefits of specific processes may be modeled. The four processes investigated here are inspections, reuse, Cleanroom software development, and agile development.

4.2.1 Benefits of Inspections

The cost of repairing a defect is cheapest if the defect is detected close to the point of its insertion. If, for example, design defects are not discovered until the test or maintenance phase of a project, the cost to repair the defect is significantly greater than if the defect is discovered during the design phase. The objective, and thus the benefit, of formal inspections is to find defects at or near their points of insertion.

Table 4.4 defines an anticipated defect rate of 7 defects/KSLOC for 39.967 KSLOC which results in 280 estimated defects. Table 4.4 computes and compares the total rework costs caused by defects being discovered and repaired following a formal inspection process or by following an informal inspection process. Total rework costs are computed based on three parameters

- The total number of defects detected in phase i , d_i
- The amount of time, in hours, to detect an error in phase i , t_i

- The average hourly labor rate to fix an error, R

The total rework costs, RC, is defined in terms of these parameters:

$$RC = R \bullet \sum d_i t_i$$

Table 4.4 computes defects detected by phase and rework hours in each phase. As stated by Curtis (1995) and depicted in Table 4.4, 35% of defects induced into a system occur during the design of the software and 65% during the coding phase. Table 4.4 also shows that, as described by Jones (1996), formal inspections detect 65% of design defects and informal inspections detect 40% of design defects. Assuming equal amount of rework per defect between formal and informal inspections, in total, more rework occurs in formal inspections; however, more defects remain with informal inspections for the next phase. Jones (1996) has stated that formal code inspections remove 70% of all defects remaining, whereas code walkthroughs remove 35% of defects remaining. More defects are found by the formal method than the informal method. Significantly fewer defects remain in the software for detection during the test stage following the formal methodology. Curtis (1995) observes that defects found during the test phase are 10X as expensive to correct as during the earlier stages. A significant labor savings thus occurs with the formal methodology. Jones (1996) has stated that the average organization removes 85% of the defects before customer release, whereas organizations utilizing inspections typically remove 95% or more of the defects prior to release. This effect is also modeled in Table 4.4. Curtis (1995) stated that repairing defects in a released product costs anywhere from 80X to 100X as much as at the time of their insertion.

Another benefit of inspections is that product developers and designers attend inspections of their products. Developers learn that certain types of errors tend to occur repeatedly in their products. As a result those types of errors do not appear again and the products become more error free at the time of inspection. The average number of defects per KSLOC will thus decrease.

McCann (2004) presents a detailed cost model for evaluating the effectiveness of code inspections. In McCann's model, inspection costs consist of inspection fixed cost, preparation and meeting costs, and rework costs when defects are found. The benefits from inspections are reductions of regression testing fixed costs and test rework and regression costs. Based on

simulated data, McCann concluded that for inspections the benefits almost always outweigh the costs, and ROI factors of up to 20 can be realized.

El Emam (2005) has developed a comprehensive ROI model which includes the effectiveness of inspections. For both design and code inspections, the effectiveness rate is 57%; this percentage of defects present in design or code that are found. Furthermore, El Emam estimates the effort to find and correct a defect during design or code inspections is 1.5 person-hours compared to 6 person-hours during testing. El Emam's model is based on extensive data from industry, so the positive ROI from using inspections appears to be valid for a wide range of programs.

4.2.2 Modeling the Effects of Reuse

Software reuse provides two cost benefits to the software manager. First, reusing code means that the reused product does not have to be developed and thus less effort and cost is required. Second, reused code has less defects/KSLOC than new code.

These double effects are shown in Table 4.5. This table shows four identical products with varying levels of reuse, from 0% to 90%. Lim (1994) states that reused code requires between 10% and 63% of the development effort as new code. For purposes of this example, 30% has been selected as the equivalent ratio. As can be seen on the line titled "Equivalent Cost", $1.6 million in development costs can be saved by improving reuse to a 90% level. As stated by Card (1994), successful reuse programs have achieved a 30 to 80% reuse level. Reuse at the levels shown in Table 4.5 is thus possible.

Phase	Formal Inspections	Informal Inspections
Design		
% Defects Introduced	35%	35%
Total Defects Introduced	98 Defects	98 Defects
% Defects Detected	65%	40%
Defects Detected	64 Defects	39 Defects
Rework Hours/Defect	2.9 Staff Hours	2.9 Staff Hours
Total Design Rework	**185 Staff Hours**	**114 Staff Hours**
Coding		
% Defects Introduced	65%	65%
Total Defects Introduced	182 Defects	182 Defects
% Defects Detected	70%	35%
Defects Detected	151 Defects	84 Defects
Rework Hours/Defect	2.9 Staff Hours	2.9 Staff Hours
Total Coding Rework	**440 Staff Hours**	**245 Staff Hours**
Test		
Defects Found in Test	51 Defects	114 Defects
Rework Hours/Defect	29.1 Staff Hours	29.1 Staff Hours
Total Test Rework	**1478 Staff Hours**	**3326 Staff Hours**
% of Defects Removed	95%	85%
Maintenance		
Defects Left for Customer	14 Defects	42 Defects
Post Release Defects/KSLOC	0.35 Defects/KSLOC	1.05 Defects/KSLOC
Rework Hours/Defect	290.6 Staff Hours	290.6 Staff Hours
Total Maintenance Rework	**4066 Staff Hours**	**12197 Staff Hours**
Totals		
Total Rework	**6168 Staff Hours**	**15881 Staff Hours**
Total Rework Costs	**$240,560**	**$619,369**
Total Savings	**$378,809**	

Table 4.4: Effects of Inspections on Rework

In the block titled "Estimated Maintenance Costs", the impact of less rework on reused code is shown. When combined with lower development effort, increasing the levels of reuse results in a $1.8 million (68%) cost savings. Rework costs are estimated in a similar fashion to Table 4.5. However, reused code has been observed by Lim (1994) to have less than 1 error/KSLOC. This impact on the reused code is taken into consideration in computing rework costs.

	Without Reuse	With Reuse	With Reuse	With Reuse
Estimated SLOC	39,967 LOC	39,967 LOC	39,967 LOC	39,967 LOC
% Reuse	0%	30%	60%	90%
Equivalent Ratio on Reuse	30%	30%	30%	30%
Equivalent Code	39,967 LOC	31,574 LOC	23,181 LOC	14,788 LOC
Cocomo Effort Estimate	487 Staff Months	374 Staff Months	265 Staff Months	160 Staff Months
Equivalent Cost	$2,886,543	$2,216,769	$1,568,258	$947,903
Schedule Length	27 Calendar Months	24 Calendar Months	21 Calendar Months	18 Calendar Months
Estimated Rework				
New Code	$240,560	$168,392	$96,224	$24,056
Reused Code	$0	$10,310	$20,619	$30,929
Total Rework	$240,560	$178,702	$116,843	$54,985
Estimated Maintenance Co	$158,560	$117,788	$77,015	$36,242
Development Effort + Mainte	$3,045,104	$2,334,557	$1,645,273	$984,146
Savings of Reuse over No Reuse		$710,547	$1,399,831	$2,060,958
% Reduction		23%	46%	68%

Table 4.5: The Effects of Reuse on Development Effort and Rework

Jones (2000) has shown that achieving significant reuse can not be achieved until Stage 5 of process improvement cycle. This means that achieving significant levels of reuse can not occur until a stable software development process is achieved and that could take many months. For example, in Table 4.5 significant levels of quality reuse will not occur until month 20. The benefits of reuse can not be achieved until reusable software is available. Reusable software costs more to develop than traditional software, because an additional design constraint - making the software reusable - is added to the software requirements. Lim (1994) has observed that development of reusable software costs between 111% and 200% of traditional software.

Boehm, et al (2000) shows the effects of reuse in the now-famous COCOMO II software cost estimating model by calculating an "equivalent size" in thousands of source lines of code (KSLOC) from reuse. According to Boehm (2000), adapted code is preexisting code that is treated as "white box" and is modified for use with a product. Variants of the two reuse equations are:

1) Equivalent KSLOC = Adapted KSLOC x (AA + AAF + (SU x UNFM) / 100) for AAF >50
2) Equivalent KSLOC = Adapted KSLOC x (AA + AAF x (1 + .02 x SU x UNFM) / 100)
   For AAF <=50

Where,

AA = Assessment and Assimilation increment, from 0 to 8
AAF = Adaptation Adjustment Factor (AAF = 0.4 x DM + 0.3 x CM + 0.3 x IM)
SU = Software Understanding increments, from 10 to 50
UNFM = Programmer unfamiliarity with software, from 0 to 1
CM = Percent code modified
DM = Percent design modified
IM = Percent Integration required for the adapted software
Adapted KSLOC = Adapted code, in thousands of source lines of code.

These equations show that reuse effort is not linearly related to AAF, and even "minor" modifications can require considerable effort if values of AA, SU, and UNFM are relatively high. If the software is not modified, as is the case of COTS software, DM and IM are 0, and SU and UNFM are also 0, so savings are generally more pronounced.

Jensen (2004) presents a simplified equation for assessing the costs to develop a reusable component and the savings from using this component. Jensen's equation is:

$$C = (1 - R) + (b + a / n) \times R$$

Where C is the reuse cost factor, R is the portion of the system to have the reused code, b is the relative cost of incorporating the reused code into the system, a is the relative cost to develop a reusable component (which usually is between 1 and 2), and n is the number of times the component is reused. As long as (b + a/n) is less than 1, C will be less than 1, and there will be a positive cost incentive. Of course, if the component has already been developed, the reuse will be more positive. This does show, however, that developing software for future reuse is more costly than developing software without reuse considerations, and the savings do not occur until the software is actually reused. In the COCOMO II model, a reuse parameter (RUSE) captures the effort to develop for future reuse. This can add 24% to software development effort if the software program is to be used across multiple product lines.

An extension of reuse which is gaining increased attention is software product lines. Boehm, Brown, et al (2004) have developed a Constructive Product Line Investment Model (COPLIMO) to estimate the cost benefits of product lines. COPLIMO is an extension of

Boehm's COCOMO II software cost estimating model (Boehm, 2000). In this model, the cost of developing (or writing) software for product line reuse is

PMR = PMNR x (PFRAC + RCWR x (AFRAC + RFRAC))

Where,
PMR = Person-months (of effort) for reused software
PMNR = Person-months if software will not be reused (the nominal case)
PFRAC = Product-specific fraction of software size (typically 0.4)
AFRAC = Adapted fraction of software size to make product work well (typically 0.3)
RFRAC = Reused fraction of software size, usable as a "black box" (typically 0.3)
RCWR = A product of increased COCOMO II effort multipliers for reuse (RUSE), reliability (RELY),and documentation (DOCU) required for product line reuse (typically 1.5 to 2.0)

The added effort for future reuse will typically be 40% to 60% of the development effort without reuse considerations. However, the savings are realized in a decrease of equivalent KSLOC for future product line software programs. The ROI breakeven point is usually 2 to 3 future programs.

Bockle, et al (2004) describe an economic model for calculating ROI for establishing product lines. Their general equation for establishing and producing a product line of *n* products is:

$C\ (Org) + C\ (Cab) + \Sigma\ C\ (Unique\ P_i) + \Sigma\ C\ (Reuse\ P_i)$

while the cost of producing *n* individual products is

$\Sigma\ C\ (Prod)$

Where,
C (Org) = Cost of an organization to adopt a product line approach
C (Cab) = Cost to develop a core asset base for a product line being built
Σ C (Unique P_i) = Sum of costs for product unique software for *n* products.
Σ C (Reuse P_i) = Sum of costs to reuse core assets for *n* products.
Σ C (Prod) = Sum of costs of developing *n* individual products.

ROI is the cost savings divided by the cost of investment, or (Cost of old way – cost of new way) divided by (C(Org) + C (Cab)).

In the example presented, where an organization has developed a set of independent products and wants to look at the benefits of using a product line approach where, in person-months, C (Prod) = 12, C (Org) = 2.4, C (Cab) = 13, C (Unique) = 0.72, and C (Reuse) = 0.84,

the ROI is – 34.3% for 1 product, but is + 38.1% for 2 products, and increases to 618.0% for 10 products. Of course, actual program values may vary, but this does show that a positive ROI can occur after as few as 2 new programs are developed.

According to Galorath (2007), the effective size of the existing software can be determined using the formula:

Effective size = existing size × (0.4 × redesign % + 0.25 × reimplementation % + 0.35 × retest %).

This equation is similar to Boehm's AAF discussed above, but computing the percentages is elaborated in Table 4.6. In most cases, the effective size with reuse will be less than the size of an equivalent program being developed from scratch, which results in lower cost and a shorter schedule.

4.2.3 Modeling the Effects of Cleanroom Software Development

Hausler (1994) observes that the Cleanroom methodology increases productivity between 1.5X and 5X. Sherer (1996) has observed a 3.62X productivity increase with the Cleanroom methodology. This productivity increase effect is modeled in Table 4.7 through the equivalent ratio factor, utilizing the value as reported by Sherer. In this table, three different approaches are examined: with Cleanroom methods, traditional methods with formal inspections, and traditional methods with informal inspections (walk-throughs).

Redesign Breakdown		
	Formula to compute redesign percentage:	0.22 x A x 0.78 x B + 0.5 x C + 0.3 x (1-(0.22 x A + 0.78 x B) x (3 x D x E)/4
Weight	***Redesign Component***	***Definitions***
0.22	Architectural design change (A)	Percentage of preexisting software requiring architectural design change
0.78	Detailed design change (B)	Percentage of preexisting software Requiring detailed design change
0.5	Reverse engineering required (C)	Percentage of preexisting software not familiar to developers; requires understanding and/or reverse engineering to achieve modification
0.225	Re-documentation required (D)	Percentage of preexisting software requiring design re-documentation
0.075	Revalidation required (E)	Percentage of preexisting software requiring revalidation with new design
Reimplementation Breakdown		
	Formula to compute re-implementation percentage:	0.37 x F + 0.11 x G + 0.52 x H
Weight	***Reimplementation Component***	***Definitions***
0.37	Recoding required (F)	Percentage of preexisting software requiring actual code changes
0.11	Code review required (G)	Percentage of preexisting software needing code reviews
0.52	Unit testing required (H)	Percentage of preexisting software requiring unit testing
Retest Breakdown		
	Formula to compute retest percentage:	0.10 x J + 0.04 x K + 0.13 x L + 0.25 x M + 0.36 x N + 0.12 x P
Weight	***Retest component***	***Definitions***
0.10	Test plans required (J)	Percentage requiring test plans to be rewritten
0.04	Test procedures required (K)	Percentage requiring test procedures to be identified and written
0.13	Test reports required (L)	Percentage requiring documented test reports
0.25	Test drivers required (M)	Percentage requiring test drivers and simulators to be rewritten
0.36	Integration testing (N)	Percentage requiring integration testing
0.12	Formal testing (P)	Percentage requiring formal demonstration testing

Table 4.6: Redesign, Reimplementation, and retest Breakdown

Like software inspections, the Cleanroom methodology also results in lower rework and maintenance costs. The impact of this is shown in Table 4.8. The first effect to be noticed is that Basili (1994) has observed that the average number of defects decreases from 7 to 5 defects/KSLOC. Fewer defects are thus induced into the software. This spreadsheet model also demonstrates, as shown by Linger (1993), that software typically enters the test phase with near zero defects. The costs for testing and maintenance are thus significantly reduced.

	Cleanroom Methodology	Formal Inspections	Informal Inspections
Estimated SLOC	39,967 LOC	39,967 LOC	39,967 LOC
Equivalent Ratio	22%	100%	100%
Equivalent Code	8,651 LOC	39,967 LOC	39,967 LOC
Effort Estimate	75 Staff Months	419 Staff Months	419 Staff Months
Equivalent Cost	$447,175	$2,482,427	$2,482,427

Table 4.7: Comparison to Cleanroom Development Costs

4.2.4 Modeling the Effects of Agile Software Development

Boehm and Turner (2004) show that a parameter in the COCOMO II cost model (Boehm, et al, 2000), Architecture and Risk Resolution (RESL), may be affected by the degree of agile methods in a program. RESL is a combined measure of how well architecture is defined early in a program and how much risk reduction has taken place. For planned (versus agile) programs, RESL will be better and probably result in less rework, but will incur much up-front effort. There is a ‘sweet spot’, a balance between using planed and agile methods, which will result in the lowest development cost. This balance tends toward a greater portion of agile methods for smaller programs, and toward planned methods for large programs, especially those for which requirements remain relatively stable.

	Cleanroom Methodology	Formal Inspections	Informal Inspections
Lines of Code	39,967 LOC	39,967 LOC	39,967 LOC
Average # Defects/ KSLOC	5 Defects/ KSLOC	7 Defects/ KSLOC	7 Defects/ KSLOC
Total Defects Expected	200 Defects	280 Defects	280 Defects
Design			
% Defects Introduced	35%	35%	35%
Total Defects Introduced	70 Defects	98 Defects	98 Defects
% Defects Detected	80%	65%	40%
Defects Detected	56 Defects	64 Defects	39 Defects
Rework Hours/ Defect	2.5 Staff Hours	2.5 Staff Hours	2.5 Staff Hours
Total Design Rework	140 Staff Hours	159 Staff Hours	98 Staff Hours
Coding			
% Defects Introduced	65%	65%	65%
Total Defects Introduced	130 Defects	182 Defects	182 Defects
% Defects Detected	98%	70%	35%
Defects Detected	141 Defects	151 Defects	84 Defects
Rework Hours/ Defect	2.5 Staff Hours	2.5 Staff Hours	2.5 Staff Hours
Total Coding Rework	353 Staff Hours	378 Staff Hours	211 Staff Hours
Test			
Defects Found In Test	1 Defects	51 Defects	114 Defects
Rework Hours/ Defect	25.0 Staff Hours	25.0 Staff Hours	25.0 Staff Hours
Total Test Rework	22 Staff Hours	1271 Staff Hours	2861 Staff Hours
% of Defects Removed	99%	95%	85%
Maintenance			
Defects Left for Customer	2 Defects	14 Defects	42 Defects
Post Release Defects/ KSLOC	0.05 Defects/ KSLOC	0.35 Defects/ KSLOC	1.05 Defects/ KSLOC
Rework Hours/ Defect	250.0 Staff Hours	250.0 Staff Hours	250.0 Staff Hours
Total Maintenance Rework	500 Staff Hours	3497 Staff Hours	10491 Staff Hours
Maintenance $	$19,484	$136,386	$409,159
Totals			
Total Rework	1014 Staff Hours	5306 Staff Hours	13660 Staff Hours
Total Rework Costs	$39,544	$206,918	$532,752
Effort $ + Maintenance $	$466,659	$2,618,814	$2,891,586
$ Improvement for Cleanroom		$2,152,155	$2,424,927
% Improvement Over Cleanroom		82%	84%

Table 4.8: Rework and Maintenance Costs in Cleanroom

Boehm and Turner (2004) also examined the results of pair programming, a characteristic of XP and agile development. Pair programming tended to increase effort, but reduce schedule and defect rate. One possible consequence of pair programming is improved morale and

teamwork, which are accounted for in several COCOMO II parameters including personnel continuity (PCON) and team cohesion (TEAM). Higher ratings for these parameters result in reduced effort and schedule.

Jones (2007) states that about 15% of the current programs of less than 1,000 function points (or about 50,000 lines of Java code) are using agile methods. He has presented the following rules of thumb for agile projects. Some of the metrics are based on an "average" size agile program which is 500 function points, or about 25,000 lines of Java code:

- Agile projects start with a two-week planning period.
- Five increments of 100 function points, one every two weeks.
- Agile projects start with a two-week planning period.
- Typical schedules in months are function points raised to the 0.33 power.
    - For 500 function points, the schedule will be 7.7 months.
- A typical agile assignment scope is 100 function points.
- Agile teams usually consist of five people.
- The average agile production rate is 36 function points per month.
- Defect potentials are about 3.5 per function point.
- Defect removal efficiency (defects removed before deployment) is about 92%.
- The bad fix injection rate is about 2%.
- Delivered defects average 0.34 per function point.
- The average agile program, once deployed, can be maintained by a single programmer.

Jones also provides some rules of thumb for XP and Scrum, which are agile development methods.

Of course, any rules of thumb should be used with caution since they only represent averages and individual programs often vary considerably from averages; however, these averages can be used to obtain a coarse estimate for a new agile program. Also, the rules of

thumb show that agile methods are usually superior to traditional methods in areas such as schedules, staffing required and delivered defects.

4.3 Modeling Secondary Benefits of Process Improvements

The cost impact of many secondary benefits of software process improvement can be estimated. This Section will examine the cost impact of improved schedules, improved staff retention and turnover, customer satisfaction, and reduced risk.

4.3.1 Cost Benefit of Improved Schedules

In contract software development work, contracts may be negotiated with bonuses and penalties built in to the contract based on an assumed delivery date. Given an increased ability to deliver ahead of schedule with process improvements, organizations could benefit financially from bidding on award fee or bonus-type contracts. If, however, an organization is typically late with deliverables, and incur penalties, comparing this history with the likely bonuses to be achieved with the improvements will be to a software development organization's advantage. Table 4.9 shows the Penalties/Bonuses Anticipated without Improvements based on past projects and the Penalties/Bonuses Anticipated with Improvements for future projects. Bonuses and penalties may be based on the total value of the contract. Diaz and Sligo (1997) state that finishing projects early also allows their company (a DoD contractor) to apply more resources to obtaining more business. More business would increase project backlog, allow us to hire more people, and increase revenues and profits.

	Without Improvement	With Improvement
Schedule Length (estimated	23 Calendar Months	18 Calendar Months
Schedule Reduction		5 Calendar Months
(Penalty)/Bonus Anticipated	($50,000)	$50,000

Table 4.9: Cost Impact of Early Delivery in Contract Organization

In commercial organizations, product sales and profits are typically modeled and forecast by sales groups or some other group based on the anticipated demand for products beginning from a given product release date. Shipping a quality product many months prior to the originally scheduled release date may result in being the first to the market with this product or potentially increase sales volume because of the early release. This spreadsheet will have to be

discussed with the sales organization to establish likely values. This analysis is shown in Table 4.10.

	Without Improvement	With Improvement
Schedule Length (estimated	23 Calendar Months	18 Calendar Months
Schedule Reduction		5 Calendar Months
Projected Sales	$10,000,000	$10,500,000
Additional Sales		$500,000

Table 4.10: Cost Impact of Early Delivery in Commercial Organization

Most software estimating models show there is a direct relationship between cost and schedule such that, as one increases or decreases, the other will also. In the original COCOMO model (Boehm, 1981), the schedule equation for embedded software systems is:

$$TDEV = 2.5\ (PM)^{0.32}$$

Where TDEV is development time in months and PM is effort in person-months.

The COCOMO II model (Boehm, et al, 2000) has more complex scheduling equations, but there is still a direct relationship between schedule and effort. It should be evident, then, that SPI activities that reduce effort will also reduce schedule. One COCOMO II example is process maturity (PMAT), a parameter which has an exponential effect on effort. PMAT reflects a company's CMM level (discussed earlier) with a "Very Low" rating for CMM Level 1 to "Extra High" for CMM Level 5. The corresponding numerical values range from 7.80 for Very Low to 0.00 for Very High. Higher numerical values result in more effort, so a higher CMM rating (or CMMI rating) will result in relatively lower effort and schedule. Also in COCOMO II, reuse usually reduces effective software size. Since there is a direct relationship between size and effort, reuse tends to reduce effort and, in turn, schedule.

There are times, however, when practices which reduce schedules may result in increased effort. As reported by Boehm and Turner (1984), pair programming, an agile development method, has usually resulted in reduced schedules but increased effort due to an "extra" person being needed. The studies reported, however, were done on very small projects and with little "jelling" activity beforehand. If pair "jelling" is done beforehand, the increase in effort tends to be sharply reduced or even eliminated. Furthermore, pair programming (like inspections) reduces defect rates, which can result in cost savings later in a program.

4.3.2 Cost Benefit of Reduced Staff Turnover and Better Staff Retention

It is important to have a highly qualified staff within a software development organization. Yamamura and Wigle (1997) document that employee satisfaction grew from 74%

to 96% satisfied because of their improvement efforts. Broadman and Johnson (1995) observe that process improvement results in improved morale and less employee turnover. Curtis (1995) has identified five cost factors that relate to turnover costs: recruiting costs, relocation costs, training costs, lost performance until person is replaced, and lower productivity per day from new employee. Based on the projects being affected, the loss of key personnel could have a negative impact on the personnel qualification factors within the COCOMO model, resulting in increased development costs and schedule lengths. Several analyses are proposed below to address turnover costs.

The first analysis relates to company costs that are incurred when personnel leave and new personnel need to be hired. The number of employees that leave a company (or stay with a company) is related to employee satisfaction. The proposed metrics to be analyzed are Yearly Turnover Costs Without Improvement and Yearly Turnover Costs With Improvement, and are computed based on the spreadsheet in Table 4.11:

Metric	Without Improvement	With Improvement
Current # Software Eng.	300 employees	300 employees
Employee Satisfaction	74%	96% (est.)
Turnover Ratio	10%	2% (est.)
Number to be Replaced/Year	30 employees	5 employees
Recruiting $/Replaced Emp.	$2,500	$2,500
Relocation $/Replaced Emp.	$15,000	$15,000
Training $/Replaced Emp.	$3,000	$3,000
Average Months to Replace	4 Months	4 Months
Total Recruiting Costs	$75,000	$12,500
Total Relocation Costs	$450,000	$75,000
Total Training Costs	$90,000	$15,000
Yearly Turnover Costs	$615,000	$102,500
Savings in Turnover Costs		$512,500

Table 4.11: The Cost Impact of Staff Turnover

In this table, current employee satisfaction and turnover ratios may be available from an organization's Human Resources department. For purposes of this example, estimated satisfaction ratios with improvement assume proportional improvements to those noted by Yamamura and Wigle (1997). Estimated turnover ratios after improvement are assumed to be inversely proportional to employee satisfaction ratios. Historical employee satisfaction and turnover ratios within an organization should be studied to understand the relationship with employee satisfaction. The turnover ratio is multiplied by the current number of employees to compute the "Number to be Replaced/Year." Average Months to Replace an employee, and

Recruiting, Relocation and Training costs per replaced employee are based on historical data. The "Number to be Replaced/Year" is multiplied by average recruiting, relocation, and training costs, and these are summed to compute the Yearly Turnover Costs.

Since key technical contributors to an organization's projects may be the most heavily recruited by outside companies, the cost of the loss of key contributors should be evaluated. Since employee satisfaction increases in organizations that employ process improvement techniques, the probability that key personnel will leave will decrease. The COCOMO Model as developed in the earlier spreadsheet is used to evaluate the cost and schedule impact on projects by adjusting COCOMO personnel adjustment factors. The values to be compared are Development Costs Without the Best Person, or Key-Employee (Without Improvement), Development Costs With the Best Person (With Improvement), Schedule Length Without the Best Person (Without Improvement), and Schedule Length With the Best Person (With Improvement), as shown in Table 4.12.

	Without Improvement	**With Improvement**
Project DRAT	Without the Best Person	With the Best Person
Cocomo Personnel Attributes		
ACAP - Analyst Capability	NOMINAL	HIGH
AEXP - Analyst Experience	NOMINAL	HIGH
PCAP - Programmer Capab	LOW	LOW
VEXP - Team Exp. w/VM	LOW	LOW
LEXP - Team Exp. w/HOL	NOMINAL	HIGH
Development Costs	$2.9M	$2.2M
Development Cost Impact		$0.7M Less
Schedule Length	27 Calendar Months	24 Calendar Months
Schedule Length Impact		3 Months Less

Table 4.12: The Value of Key Contributors

In Table 4.12, the values assigned to ACAP, AEXP, PCAP, VEXP, and LEXP are standard COCOMO 1.1 definitions (Boehm, 1981) and are established by evaluating the personnel capabilities on each project with and without that key individual. Development costs and schedule lengths are computed based on multiplying COCOMO labor estimates with average hourly labor rates. The Development Cost Impact row subtracts Development Costs With Key-Employee from Development Costs Without Key-Employee. It highlights the value this employee brings to the project. Similarly, the Schedule Length Impact row subtracts Schedule

Length With Key-Employee from are Schedule Length Without Key-Employee**.** It highlights the impact this individual has on delivering product earlier.

COCOMO II (Boehm, et al, 2000) has added two more personnel parameters, Team Cohesion (TEAM) and Personnel Continuity (PCON). TEAM (like PMAT discussed earlier) has an exponential effect on effort (and schedule), and reflects the ability of the development team to interact. The ratings range from "Very Low" for teams with very difficult interactions to "Extra High" for seamless interactions. The corresponding numerical values range from 5.48 for Very Low to 0.00 for Extra High. Higher numerical values result in more effort, so having a cohesive team will result in relatively lower effort and schedule. For PCON, a linear effort multiplier, ratings range from "Very Low" for a high turnover rate of 48% per year to "Very High" for a low turnover rate of 3% per year. The corresponding numerical values range from 1.29 for Very Low to 0.81 for Very High. Higher numerical values result in more effort, so having a lower turnover rate will result in relatively lower effort and schedule. Usually, SPI practices will result in higher morale, higher team cohesion, and lower turnover rates.

4.3.3 Cost Benefit of Improved Customer Satisfaction

Although mentioned as a benefit of process improvement by others (e.g., Diaz and Sligo (1997), Brodman and Johnson (1995)), no specific repeat business dollar values have been attributed to process improvement in the literature. To establish the Repeat Business analysis shown in Table 4.13, other projects within an organization will have to be investigated, examining post-release defects, schedule performance, and dollar value of new work received from customers. Alternatively, the repeat business achieved by competitors who use more modern methods, with higher software quality, and better cycle times could be evaluated to develop an estimate of the dollar value of Repeat Business with and without improvement.

	Without Improvement	With Improvement
Repeat Business	$1,000,000	$5,000,000
Additional Business		$4,000,000

Table 4.13: Repeat Business

Ceschi, et al (2005) studied the effects of agile development, an SPI activity, on customer satisfaction. This area is important because, based on a survey of more than 8,000 projects, five of the top six reasons for failure stem from communication problems between the development team and the customer. Agile methods help alleviate these problems because customers are

usually on-site and directly available for feedback. For programs studied, 90% of the customers of agile programs were either satisfied or very satisfied while only 70% of the customers for traditional plan-based projects were satisfied or very satisfied. This is an added benefit to using agile development methods, which show a positive ROI in many instances.

Boehm, Huang, et al (2004) have developed the Information Dependability Attribute Value estimation (iDAVE) model to estimate the ROI of software dependability. iDAVE uses many of the parameters from the COCOMO II model (Boehm, et al, 2000) and from the Constructive Quality Model (COQUALMO), an extension of the COCOMO II model, for software quality (Boehm, et al, 2000). One of the key COCOMO II parameters used in iDAVE is required reliability (RELY), which is rated from very low to very high depending on the consequences of a software failure. The ratings (and numerical values) are: very low (0.82) for a slight inconvenience, low (0.92) for low easily recoverable losses, nominal (1.00) for moderate easily recoverable losses, high (1.10) for high financial losses, and very high (1.26) for risk to human life. In a case study for a commercial application, raising RELY from nominal to high would add $344,000 to a $3,440,000 program, but the results of the final software product would be an increase in mean-time-between-failure from 300 hours to 10,000 hours and a resultant decrease in losses over a 5-year period due to downtime of $5,150,000, for a positive ROI of 14% per year. Additionally, a customer satisfaction rating on a scale from 0 to 5 would increase from the current 1.7 to 4.6 five years later due to fewer late deliveries, greater ease of use, and more in-transit visibility.

According to El Emam (2005), assessing the ROI from customer satisfaction is challenging because few software companies collect customer satisfaction data, and because customer satisfaction usually involves more than a product itself, such as service and vendor reputation. Still, other research has shown that some SPI practices can result in increased customer satisfaction and a positive ROI.

4.3.4 Cost Benefit of Reduced Risk on Software Projects

The risks and potential cost impacts of improving the software organization can be compared with the risks and potential cost impacts of not improving. Strassman (1990) makes the following three statements which suggest the importance, from a management perspective, of risk analysis: "By making the risks of technology more explicit, you create a framework for diagnosing, understanding and containing the inherent difficulties associated with technological and organizational innovation," "The best way to avoid failure is to anticipate it," and "Risk analysis is the correct analytical technique with which one can examine the uncertainty of

Information Technology investments prior to implementation." Dorofee (1997), in discussing lessons learned from applying the SEI's risk management program, states that written risks are harder to ignore than verbal concerns. He suggests that a risk information sheet, like the one described below, be used to document risks.

Especially within the DoD, use (reuse) of COTS software is viewed by many as a silver-bullet to reduce the escalating costs of software development, reduce cycle time, and improve quality. As noted by Carney and Oberndorf (1997), use of COTS products may be beneficial or it may cause greater problems. The authors noted many risks that need to be considered in selecting COTS products. Those risks are summarized in the "Risk Description" column of Table 4.14, the following risk assessment spreadsheet:

Risk Number	Risk Description	Potential Cost	Likelihood	Weighted Cost
6	Product Not Available for life of my product.	$2,500,000	HIGH	$1,875,000
10	COTS Does Not Integrate With Other COTS.	$2,000,000	MEDIUM	$500,000
5	COTS Products Has Critical Bugs.	$5,000,000	LOW	$250,000
8	COTS Vendor Goes Out Of Business.	$5,000,000	LOW	$250,000
1	Improperly Planned Use of COTS.	$2,500,000	LOW	$125,000
11	Improper Market Research of COTS.	$2,500,000	LOW	$125,000
7	Vendor Stops Support of Product.	$2,000,000	LOW	$100,000
9	Future Incompatibilities with Hardware.	$2,000,000	LOW	$100,000
3	COTS Don't Meet Requirements.	$1,250,000	LOW	$62,500
4	COTS Not Plug-And-Play.	$1,250,000	LOW	$62,500
2	COTS will need modification to work.	$200,000	MEDIUM	$50,000

Table 4.14: Risks of COTS Reuse

The values in the columns of Table 4.14 are merely exemplary. Each risk is assigned a number and is placed in the column titled "Risk Number". The "Potential Cost" column defines the maximum impact should the risk actually occur. The probability of the risk happening is defined in the "Likelihood" column, where the values HIGH, MEDIUM and LOW are actually assigned a value between 0 and 1. The Weighted Cost column is computed by multiplying the "Potential Cost" column by the "Likelihood" column.

Since any change has risks associated with it, other risks of process improvement are shown in Table 4.15 below. These risks are failures to achieve the expectations we have for our process improvement efforts.

Risk Number	Risk Description	Potential Cost	Likelihood	Weighted Cost
2	Poor Training in New Methods	$1,000,000	LOW	$50,000
6	Unable to Achieve Rework Reductions	$900,000	LOW	$45,000
4	Inability to Achieve Productivity Goals	$750,000	LOW	$37,500
5	Unable to Achieve Cycle Time Improvements	$500,000	LOW	$25,000
3	Inability of Staff to Change	$250,000	LOW	$12,500
1	Inability to Get Management Support	$100,000	LOW	$5,000

Table 4.15: Software Process Improvement Risks

Many risks are associated with not improving, including loss of key employees, increased staff turnover, cost overruns, and late delivery of product. The impacts of these risks are summarized in the risk assessment spreadsheet in Table 4.16.

Risk Number	Risk Description	Potential Cost	Likelihood	Weighted Cost
1	Loss of Key Person #2	$700,000	MEDIUM	$175,000
2	Loss of Key Person #1	$1,000,000	MEDIUM	$250,000
3	Higher Turnover	$512,500	MEDIUM	$128,125
6	Loss of Market Leadership	$500,000	MEDIUM	$125,000
7	Loss of Repeat Business	$4,000,000	MEDIUM	$1,000,000
4	Cost Overruns	$500,000	HIGH	$375,000
5	No Award Fees	$50,000	HIGH	$37,500

Table 4.16: Risk Assessment of Not Performing Improvements

Given that the number and the complexity of the risks with any proposed course of action can become large, the proposed methods for presentation of the risks are with Weighted Risk Likelihood values. For each proposed method, the Weighted Risk Likelihood value is defined as the sum of all weighted costs for each likelihood category. As an example, the Reuse of COTS Risks from Table 4.17, would be represented as follows

Likelihood	Weighted Cost Sum
HIGH	$1,875,000
MEDIUM	$550,000
LOW	$1,075,000

Table 4.17: Weighted Risk Likelihood for COTS Reuse

This data would be graphed as shown in Figure 4.1 below. Management is most likely concerned with the risks of high or medium likelihood risks. This graph will provide management an effective way to compare risks of improvements versus no improvements.

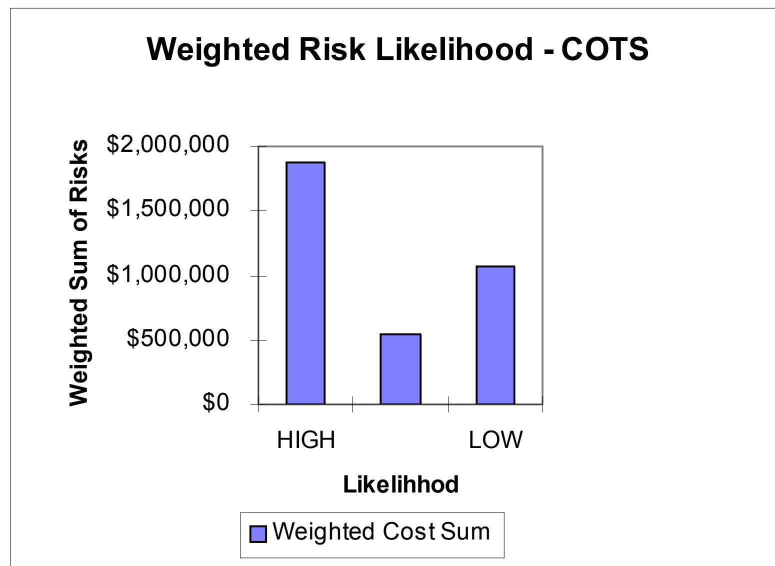

Figure 4.1: Weighted Risk Likelihood Graph for Use of COTS

In the COCOMO II model (Boehm, et al, 2000), there is a RISK value which reflects a weighted sum of schedule, product, platform, personnel, process, and reuse risks. Generally, higher values of RISK correlate with higher effort and schedule values. When the CMM level for the process maturity (PMAT) parameter (discussed earlier) increases from Level 1 to Level 5, the RISK value will decrease, as will effort and schedule. Four RISK values, schedule, process, product, and personnel risk, are affected by PMAT. Another COCOMO II parameter is TOOL, the degree of modern tool usage (which increases with SPI). Higher ratings of this parameter, like PMAT, will decrease RISK values, especially for process risk, along with effort and schedule estimates. While the architecture and risk resolution (RESL) parameter (discussed earlier) does not affect the RISK rating, it does affect effort and schedule; more risk resolution and fewer risks present will reduce effort and schedule in COCOMO II.

Huang and Boehm (2006) describe a model based on the COCOMO II and Constructive Quality Model (COQUALMO) called the Value-Based Software Quality Model. This model is an extension of the iDAVE model (Boehm, Huang, et al, 2004) discussed earlier. This model shows when to stop testing and release a product. There is a “sweet spot”, or minimum risk exposure point, which is a product of combined risk exposure of unacceptable quality and market share erosion. Too little testing will result in unacceptable quality while too much testing will result in market share erosion because of later deliveries. This model also shows that risk

exposure is less when value-based testing is used, which is a Pareto-based approach of spending more time testing components (or modules) which have the most impact on system value, instead of the traditional value-neutral testing, which treats all components equally. This model is an illustration of the emerging field of value-based software engineering.

4.4 Comparison of Results

Sections 4.2.1 through 4.2.4 address four specific software process improvement methods: Fagan Inspections, Software Reuse, Cleanroom Methodology, and Agile Development. Table 4.18 compares the development costs, rework costs, maintenance costs, SPI costs, ROI, and savings resulting for the first three methods. The column titled "Savings" computes the savings realized by subtracting "Development Costs" and "Rework Costs" of the method over the traditional method.

The cost associated with formal inspections is the cost of a 1.5 day training class in inspections. No increase in project costs was observed by any of the authors. Fagan (1986) actually saw a 25% reduction in project costs utilizing inspections and this reduction is reflected in Table 4.18.

Lim (1994) stated that for one division at HP, the costs of the reuse program were $1 million for a 55 KSLOC reuse library, or approximately 76 man-days per KSLOC. This factor is included as the costs associated with the reuse figures in Table 4.18.

Sherer (1996) shows that the costs of the Cleanroom methodology include training and coaching costs. These costs, as shown in Table 4.18, amount to approximately 17% of labor costs.

The ROI column is computed as $\frac{Savings}{Costs}$.

	Development Costs	Rework Costs	Maintenance Costs	Development + Maintenance Savings	SPI Costs	ROI
Traditional	$2,482,427	$532,657	$409,086	$0		
Formal Inspections	$1,861,821	$206,882	$136,362	$946,382	$13,212	71.63:1
Reuse						
30% Reuse	$1,906,421	$153,683	$101,297	$954,980	$199,713	4.78:1
60% Reuse	$1,348,702	$100,485	$66,233	$1,565,897	$399,426	3.92:1
90% Reuse	$815,197	$47,287	$31,168	$2,152,600	$599,139	3.59:1
Cleanroom	$447,175	$39,537	$19,480	$2,528,372	$77,361	32.68:1
Full Software Process Improvement	$670,949		$22,429	$2,344,135	$313,358	7.48:1

Table 4.18: Comparison of SPI Methods

5. Summary and Conclusions

Having previously analyzed the primary benefits of software process in the previous version of this report, and having now analyzed in this report the secondary benefits of SPI from a profit and loss perspective, improvement methods can now be compared and extensively analyzed for purposes of presentation to senior management utilizing a metrics framework. The results of the analysis for an example organization with example projects are summarized in Table 5.1. The graphical representation of the Weighted Risk Likelihood is shown in Figure 5.1.

Metric	Without Improvement	With SPI	Improvement
Primary Benefits			
Total Development Costs	$2,886,543	$780,174	$2,106,370
Total Rework Costs	$619,369	$26,080	$593,288
Average Schedule Length	27 Calendar Months	17 Calendar Months	10 Months
Post Release Defects	15% of Total Defects	<5% of Total Defects	80%
Secondary Benefits			
Projected Sales	$10,000,000	$10,500,000	$500,000
Penalties/Bonuses	($50,000)	$50,000	$100,000
Yearly Turnover Costs	$615,000	$102,500	$512,500
Repeat Business	$1,000,000	$5,000,000	$4,000,000
Cost of the Improvement		$373,000	($373,000)
Weighted Risk Likelihood			
High	$412,500	$0	
Medium	$1,678,125	$0	
Low	$0	$175,000	

Table 5.1: Comparing All the Metrics of Process Improvement

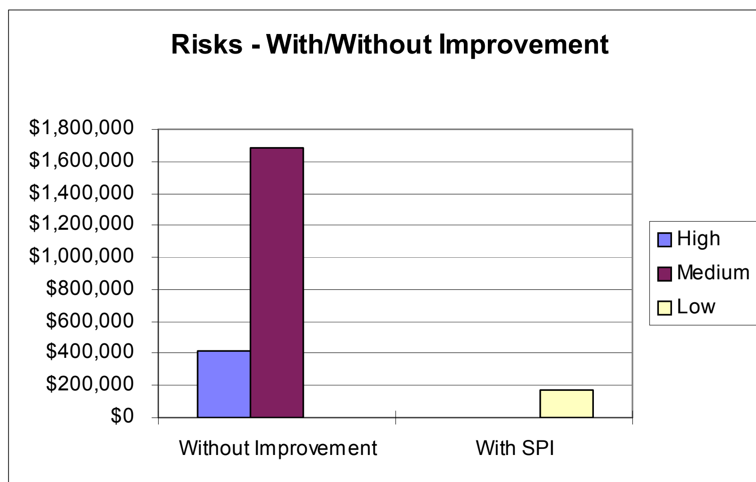

Figure 5.1 - Comparison of Weighted Risk Likelihood

5.1 The Financial Benefits of Software Process Improvement

Section 4 of this paper provides two different perspectives of the financial benefits of SPI: that of Jones (2000) as described in Section 4.1, where a complete high level model of the cost and savings impact can be developed, and that of the specific process improvements of Sections 4.2 through 4.4. It is clear from the data presented in Section 4 that SPI can have a significant bottom line cost savings to a software development organization (as much as a 67% reduction in development and rework costs).

Section 4 shows that SPI significantly:

- Reduces the amount of time and effort required to develop software
- Reduces the number of defects induced into a system
- Reduces the costs and time to find defects that are introduced
- Reduces maintenance costs on software products
- Improves productivity of the development team

Additional analysis has been performed to observe the impact on this model for various program sizes (lines of code). Figure 5.2 shows the estimate of rework costs for different process models as a function of program size. Cost savings are proportional to program size and are shown as a percentage of traditional methods in the legend. Similarly, Figure 5.3 compares development costs of traditional projects with other process models. Cost savings were found to be proportional to program size. Figure 5.4 shows the impact on schedule of various process models. Again schedule improvements were found to be proportional to program size.

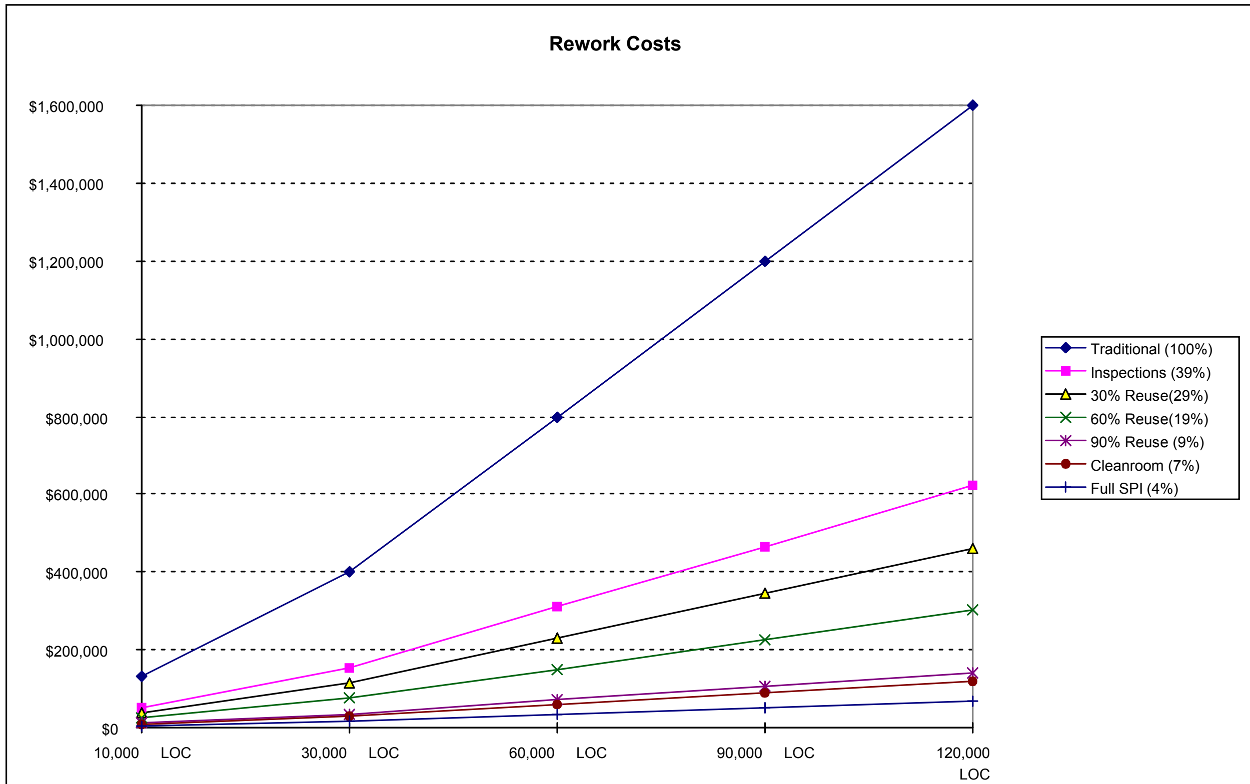

Figure 5.2: Rework as a Function of Program Size

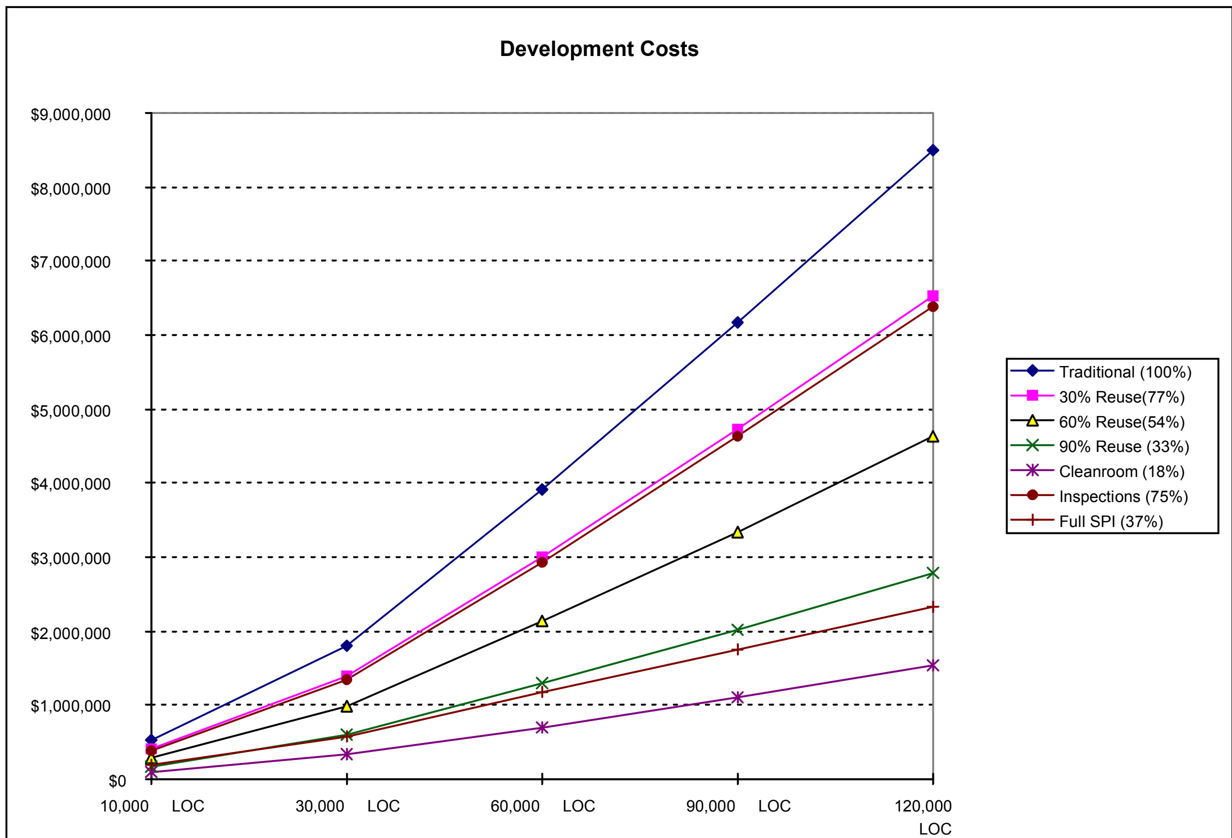

Figure 5.3: Development Costs as a Function of Program Size

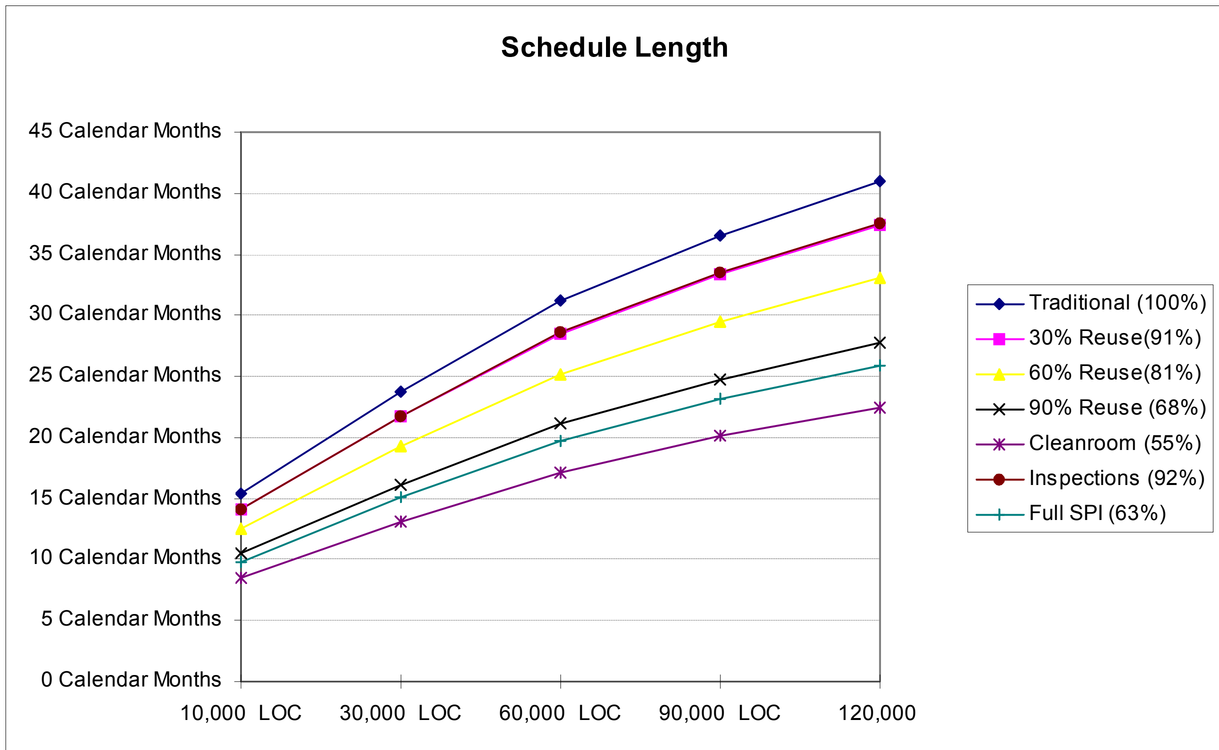

Figure 5.4: Schedule Length as a Function of Size

5.2 The Secondary Benefits of Software Process Improvement

This report has developed a financial model for assessing secondary benefits of software process improvement. Software process improvement impacts the following areas:

- **Projected Sales Without Improvement** and **Projected Sales With Improvement**. This factor would be used primarily in organizations developing commercial products and measures the increase in product sales given that products will be able to be shipped earlier. The supporting data required to estimate this may not be readily available from sales. If data is available, these metrics would provide a very meaningful measure for comparison.
- **Average Historical Penalties/Bonuses** and **Average Projected Bonus**. These metrics are only of high payoff if the contract work being performed by an organization is of the type where bonuses and award fees are rewarded for high performance and on-time delivery or where penalties are incurred for poor performance.
- **Yearly Turnover Costs Without Improvement** and **Yearly Turnover Costs With Improvement.** Since process improvement improves employee morale, turnover should dramatically. Turnover costs should be reduced accordingly. The savings from less turnover could pay for the proposed improvements alone.

- **Development Costs With Key Employees**, **Development Costs Without Key Employees, Schedule Length With Key Employees**, **Schedule Length Without Key Employees**. Further complicating the costs of turnover is the impact of the possibility of losing some of the key technical employees of an organization. The impact on an individual project could be significant.
- **Repeat Business Without Improvement** and **Repeat Business With Improvement**. Improving customer satisfaction should result in repeat business. However, until some experience with developing products using modern methods is achieved, it is very difficult to estimate these metrics.
- **Weighted Risk Likelihood Without Improvement** and **Weighted Risk Likelihood With Improvement**. These two metrics are the most interesting and potentially most powerful metrics selected, because they summarize and assign a probability to all the factors that need to be considered before selecting one method over another.

6. Annotated Bibliography

Arisholm, Erik, Gallis, Hans, Sjoberg, Dag I. K., Dyba, Tore, "Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise", *IEEE Transactions on Software Engineering*, Volume 33 No. 2, February 2007, pp. 65 - 85.

A total of 295 junior, intermediate, and senior professional Java consultants (99 individuals and 98 pairs) from 29 international consultancy companies in Norway, Sweden, and the UK were hired for one day to participate in a controlled experiment on pair programming. The subjects used professional Java tools to perform several change tasks on two alternativeJava systems with different degrees of complexity. The results of this experiment do not support the hypotheses that pair programming in general reduces the time required to solve the tasks correctly or increases the proportion of correct solutions. On the other hand, there is a significant 84 percent increase in effort to perform the tasks correctly. However, on the more complex system, the pair programmers had a 48 percent increase in the proportion of correct solutions but no significant differences in the time taken to solve the tasks correctly. For the simpler system, there was a 20 percent decrease in time taken but no significant differences in correctness. However, the moderating effect of system complexity depends on the programmer expertise of the subjects. The observed benefits of pair programming in terms of correctness on the complex system apply mainly to juniors, whereas the reductions in duration to perform the tasks correctly on the simple system apply mainly to intermediates and seniors. It is possible that the benefits of pair programming will exceed the results obtained in this experiment for larger, more complex tasks and if the pair programmers have a chance to work together over a longer period of time.

Baddoo, N., Hall, T., and Jagielska, D., “Software developer Motivation in a High Maturity Company: A Case Study”, *Software Process Improvement and Practice*, Volume 11, No. 3 (May-June 2006), pp. 219-228.

Motivation has been reported to be an important determinant of productivity and quality of work in many industries. This article explores how motivation affects development work in software engineering. Based on the experiences of an organization rated at CMM Level 5, developers who work in a high maturity organization are highly motivated.

Baheti, P., Gehringer, E., and Stotts, D., "Exploring the Efficacy of Distributed Pair Programming", Proceedings of XP/Agile Universe 2002, 2002, pp. 208 - 220.

Pair programming is one of the twelve practices of Extreme Programming (XP). Pair programming is usually performed by programmers who are *collocated*—working in front of the same monitor. But the inevitability of distributed development of software gives rise to important questions: How effective is pair programming if the pairs are not physically next to each other? What if the programmers are geographically distributed? An experiment was conducted at North Carolina State University to compare different working arrangements of student teams developing object oriented software. Teams were both collocated and in distributed environments; some teams practiced pair programming while others did not. In particular, we compared the software developed by virtual teams using distributed pair programming against collocated teams using pair programming and against virtual teams that did not employ distributed pair programming. The results of the experiment indicate that it is feasible to develop software using distributed pair programming, and that the resulting software is comparable to software developed in collocated or virtual teams (without pair programming) in terms of productivity and quality.

Basili, V., McGarry, F., Page, G., Pajerski, R., Waligora, S., Zelkowitz, M., "Software Process Improvement in the NASA Software Engineering Laboratory," Technical Report CMU/SEI-94-TR-22, Pittsburgh, Pennsylvania: Software Engineering Institute, Carnegie Mellon University, December 1994.

In this report the authors describe the software process improvement work at the NASA Software Engineering Laboratory (SEL), for which the SEL was awarded the first IEEE Computer Society Process Achievement Award.

This report describes the structure of the SEL, the SEL process improvement approach, and the experimentation and data collection process. Results of some process improvement studies are included, including the results of analyses of the Cleanroom approach and development in Ada versus Fortran.

The SEL uses the CMM model for assessing process and for selecting potential process changes. The SEL's three phase approach for analyzing potential process improvements is to understand the current state, measure the impact of improvements on products generated, and package successful improvements.

This paper is of interest here because it provides statistics of effort distribution by life cycle phase and by activity. It also provides Cleanroom productivity and quality information, and describes the benefits achieved from reuse.

Bockle, G., Clements, P., McGregor, J., Muthig, D., and Schmid, K., "Calculating ROI For Software product Lines", *IEEE Software*, Volume 21, Issue 3 (May-June 2004), pp. 23-31.

Product line engineering has become an important and widely used approach for efficiently developing portfolios of software products. The idea is to develop a set of products as a single, coherent development task from a core asset base (sometimes called a platform), a collection of artifacts specifically designed for use across a portfolio. This approach produces order-of-magnitude economic improvements compared to one-at-a-time software system development. Because the product line approach isn't limited to specific technical properties of the planned software but rather focuses on economic characteristics, high return on investment has become the anthem of the approach's protagonists. The software product line cost model can calculate the costs and benefits (and hence the ROI) that can be expected to accrue from various product line development situations. It's also straightforward and intuitive.

Boehm, B., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

This book is truly a classic in the area of software estimating. It describes the Constructive Cost Model (COCOMO), a non-proprietary model for estimating development and maintenance costs for software. The book is also an excellent resource for software management and software engineering topics in general. Especially noteworthy is his discussion of human versus material economics; the estimates from cost models such as COCOMO must be assessed in light of the impact they have on people, the environment, and other non-material issues.

Boehm, B., "Improving Software Productivity," *Computer*, Vol. 20, No. 9, September 1987, pp.43-57.

The article discusses avenues of improving productivity for both custom and mass produced software. It covers national, international and organizational trends; some of the pitfalls in defining software productivity; identifying factors that influence productivity (software value change and software productivity opportunity tree); productivity improvement steps; software productivity trends and conclusions.

This article provides interesting statistics about the cost impact of a mediocre team versus a very good team on a development project, the impact of facilities on productivity and the cost impact by phase of reuse.

Boehm, B., et al, *Software Cost Estimation with COCOMO II*, Prentice-Hall, Upper saddle River, NJ, 2000.

COCOMO II is a model to help the reader reason about the cost and schedule implications of software decisions that need to be made. This book and model address issues related to (1) evolutionary, risk driven and collaborative software processes; (2) fourth-generation languages and application generators; (3) commercial off-the-shelf (COTS) and reuse-driven software approaches; (4) fast-track software development approaches; and (5) software process maturity initiatives.

Boehm, B., "Get ready for Agile Methods – With Care", *IEEE Computer*, January, 2002, pp. 64-69.

Although many of their advocates consider the agile and plan-driven software development methods polar opposites, synthesizing the two can provide developers with a comprehensive spectrum of tools and options. Real-world examples argue for and against agile methods. Responding to change has been cited as the critical technical success factor in the Internet browser battle between Microsoft and Netscape. But over-responding to change has been cited as the source of many software disasters, such as the $3 billion overrun of the US Federal Aviation Administration's Advanced Automation System for national air traffic control. The author believes that both agile and plan-driven approaches have a responsible center and over-interpreting radical fringes. Agile and plan-driven methods both form part of the planning spectrum. Thus, while each approach has a home ground within which it performs very well, and much better than the other, a combined approach is feasible and preferable in some circumstances;

Boehm, B., and Turner, R., *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison Wesley, Boston, MA, 2004.

This book shows how seemingly opposite concepts of agility and discipline can and should be complementary. The authors illustrate the similarities and differences between agile and plan-driven methods, and show development strategies that combine the best of both

methods. They show how to find the “sweet spot” on the agility-discipline continuum for any given project.

Boehm, B., Brown, A., Madachy, R., and Yang, Y., “A Software Product Line Life Cycle Cost Estimation Model”, *Proceedings 2004 International Symposium on Empirical Software Engineering*, 19-20 August 2004, pp. 156-164 (USC-CSE-2004-517).

Most software product line cost estimation models are calibrated only to local product line data rather than to a broad range of product lines. They also underestimate the return on investment for product lines by focusing only on development vs. life-cycle savings, and by applying writing-for-reuse surcharges to the entire product rather that to the portions of the product being reused. This paper offers some insights based on the exploratory development and collaborative refinement of a software product line life cycle economics model, the Constructive Product Line Investment Model (COPLIMO) that addresses these shortfalls. COPLIMO consists of two components: a product line development cost model and an annualized post-development life cycle extension. It focuses on modeling the portions of the software that involve product-specific newly-built software, fully reused black-box product line components, and product line components that are reused with adaptation. This model is an extension built upon USC-CSE's well-calibrated, multi-parameter Constructive Cost Model (COCOMO) II, tailored down to cover the essentials of strategic software product line decision issues and available supporting data from industries.

Boehm, B., Huang, L., Jain, A., and Madachy, R., “The ROI of Software dependability: The iDAVE Model”, *IEEE Software*, Volume 21, Issue 3 (May/June 2004), pp. 54-61.

In most organizations, proposed investments in software dependability compete for limited resources with proposed investments in software and system functionality, response time, adaptability, speed of development, ease of use, and other system capabilities. The lack of good return-on-investment models for software dependability makes determining the overall business case for dependability investments difficult. So, with a weak business case, investments in software dependability and the resulting system dependability are frequently inadequate. Dependability models will need to support stakeholders in determining their desired levels for each dependability attribute and estimating the cost, value, and ROI for achieving those. At the University of Southern California, researchers have developed software cost- and quality-estimation models and value-based software engineering

processes, methods, and tools. We used these models and the value-based approach to develop an Information Dependability Attribute Value Estimation model (iDAVE) for reasoning about software dependability's ROI.

Bowers, Pamela, "The F/A-18 Advanced Weapons Lab Successfully Delivers a $120-Million Software Block Upgrade", Crosstalk, Volume 15 No. 1, January 2002, pp. 10 - 11.

As the F/A-18 Hornet becomes the Navy's nearly exclusive strike fighter, the Advanced Weapons Laboratory (AWL) steps up to the task of delivering a major software block upgrade. The software, called the 15C System Configuration Set (SCS), provides advancements that upgrade the interface between the aircraft mission systems and the aircrew. The AWL successfully delivered "real time" processing in an extremely mission critical system that pushes the technology envelope, and that requires absolute safety of flight.

Brodman, J. G., Johnson, D. L., “Return on Investment (ROI) from Software Process Improvement Measured by US Industry,” *Software Process--Improvement and Practice*, July 1995, pp. 35-47.

This paper summarizes the results of research funded under an SBIR Phase I project to determine costs and savings resulting from CMM-based software process improvement. The authors interviewed government and industry representatives and reviewed textbooks to determine a common definition for Return on Investment (ROI). Each source had a radically different perspective on ROI. The authors identified, graphically, the various categories industry representatives used to measure investment and returns.

The paper identifies several useful numerical findings about organizations involved in CMM-based SPI, including the amount of time spent at each SEI level, the use of various cost models and various costs associated with process improvement (e.g. cost of data collection, cost of fixing code defects). ROI and cost savings figures are given for SPI programs at Raytheon, Hughes and Tinker AFB.

Some of the more interesting findings are non-measurable benefits from a SPI program: improved morale of developers, increased respect for software from organizations external to software and less overtime, to name a few. In general many companies look at SPI, not from

a specific ROI financial perspective, but rather from the perspective of being more competitive, customer satisfaction and repeat business.

Brynjolfsson, E., "The Productivity Paradox of Information Technology," _Communications of the ACM_, Volume 36 #12 (December 1993), pp. 67-77.

This article examines why there is such a shortfall of evidence about productivity increases from Information Technology. Whereas productivity for the production sector has increased, the service sector has decreased with investments in Information Technology. He addresses in this article four possible explanations for this phenomenon: mismeasurement of outputs and inputs; lags due to learning and adjustment; redistribution and dissipation of profits where IT may only benefit certain areas - IT rearranges the shares without making it any bigger; mismanagement of information and technology. The author believes the major problem is due to mis-measurement. Relative to my paper, it shows the difficulty in measuring productivity.

The author suggests that productivity is the fundamental economic measure of a technology's contribution. Economists are puzzled by the productivity slowdown that began in the early 70s - there is an unplanned residual drop in productivity compared with the first half of the postwar period. This drop coincided with a rapid increase in use of IT, implying that IT investments may have been counterproductive. In that period output per production worker increased by 16.9%, whereas output per information worker decreased by 6.6% - concentrated in white collar worker and the most heavily endowed with high tech capital.

Burke, G., and Howard, W., “Knowledge Management and Process Improvement: A Union of Two Disciplines”, _Crosstalk_, Volume 18 #6 (June, 2005), On-line article.

The experience at the Federal Aviation Administration (FAA) shows that process improvement and knowledge management complement each other well. Process improvement helps the organization increase its effectiveness through continuous examination with a view to doing things better. Once processes are documented, roles and responsibilities are readily identified and associated activities are performed. Legacy processes are modified to reflect organizational changes. Knowledge management facilitates communication among organizations, increasing information sharing and utilizing process documentation. This information sharing promotes organizational unity and allows FAA headquarters and regional operations to function efficiently.

Card, D., Comer, E., "Why Do So Many Reuse Programs Fail?" *IEEE Software*, September 1994, pp. 114-115.

This short article discusses the reasons the authors believe some reuse programs have failed. They attempt to explain why some organizations are able to achieve 30-80% reuse, whereas others have failed. They believe that failed organizations treat reuse as a technology acquisition problem instead of a technology transition problem, and organizations fail to approach reuse as a business strategy. The authors believe the most important obstacles are economic and cultural.

Carney, D.J., Oberndorf, P.A., "The Commandments of COTS: Still in Search of the Promised Land," *Crosstalk*, Volume 10 #5 (May 1997), pp. 25-30.

This is a very good article that discusses the benefits and liabilities of using commercial off the shelf software as well as the causes and effects of the DoD mandates for increased usage of COTS products. Many current RFPs now include a mandate concerning the amount of COTS products that must be included. Hidden costs, such as understanding COTS products as system components; market research to find COTS products; product analyses to select among alternatives; licenses and warranties; product integration; revisions; coordination of support vendors; recovery when a vendor discontinues a product or goes out of business are discussed in detail. The ten commandments identified within the article include:

1. Do not believe in silver bullets.
2. Use the term precisely.
3. Understand the impact of COTS products on the requirements and selection process.
4. Understand COTS impact on the integration process.
5. Understand COTS impact on the testing process.
6. Realize that a COTS approach makes a system dependent on the COTS vendors.
7. Realize that maintenance is not free.
8. You are not absolved of the need to engineer the system well.
9. Just "doing COTS" is not an automatic cost saver.

10. Just "doing COTS" must be part of a large-scale paradigm shift.

Carney, D., Morris, E., and Place, P., Identifying Commercial Off-the-Shelf (COTS) Product Risks: The COTS UsageRisk Evaluation, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, September, 2003 (CMU/SEI-2003-TR-023).

The expansion in use of commercial off-the-shelf (COTS) products has been accompanied by an increase in program failures. Many of these failures have been due to a lack of familiarity with the changed approach that COTS products demand. This report describes the development of an approach to reduce the number of program failures attributable to COTS software: the COTS Usage Risk Evaluation (CURE). The origin of CURE and an overview of the method, along with detail on the materials and mechanisms used in CURE, are provided. The CURE process is outlined and the results of the evaluations that have been conducted are summarized. Finally, possible future directions for CURE are explored.

Ceschi, M., Sillitti, A., Succi, G., and De Panfilis, S., “Project Management in Plan-Based and Agile-Based Companies” *IEEE Software*, Volume 22, Issue 3 (May/June 2005), pp. 21-27.

Agile methods are a recent set of development techniques that apply a human-centered approach to software production. The agile approach aims to deliver high-quality products faster, producing satisfied customers. We conducted an empirical study to investigate whether agile methods change and improve project management practices in software companies. Survey results show that adopting agile methods appears to improve management of the development process and customer relationships. This article has given a first analysis of the advantages and disadvantages of adopting agile methods from a project management perspective.

Ciolkowski, M. and Schlemmer, M., "Studying the Effect of Pair Programming", Proceedings of ISERN 2002, October 2002.

This paper describes a case study using a realistic task within a practical course at the University of Kaiserslautern, Germany, conducted in teams of six students and comprising about 700 person-hours of total effort. Within the case study setting, weak support was found for results obtained from earlier studies. The paper describes experiences made in conducting the case study and suggests improvements for further investigations.

CMMI Product Team, *CMMI For Development, Version 1.2*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006 (CMU/SEI-TR-2006-008).

This document, which contains more than 500 pages, is the definitive work on the CMMI. It contains a wealth of information on use of the CMMI, capability and maturity levels, and, especially, the generic goals and process areas to reach higher levels of capability or maturity. It is the source document which should be referenced by anyone using or writing articles about the CMMI.

Clements, P., and Northrop, L., *Software Product Lines: Practices and Patterns*, Addison Wesley, Boston, MA, 2002.

This book is the distillation of all that the authors have learned about software product lines. They describe the essential activities, which are (1) the development of a set of core assets from which all of the products will be built, (2) the development of products using those core assets, and (3) strong technical and organizational management to launch and coordinate both core asset development and product development.

Cockburn, A., and Williams, L., "The Costs and Benefits of Pair Programming", University of Utah Technical Report, 1999.

Using interviews and controlled experiments, the authors investigated the costs and benefits of pair programming. They found that for a development time-cost increase of 15%, pair programming improves design quality, reduces defects, enhances technical skills, improves communications, and is more enjoyable at statistically significant levels.

Curtis, W., "Building a Cost-Benefit Case for Software Process Improvement," Notes from Tutorial given at the *Seventh Software Engineering Process Group Conference*, Boston, MA, May 1995.

In this tutorial the author presents methods for determining cost-benefits from software process improvement and discusses some of the published cost benefits results. He points out that differences across organizations make direct comparison of improvement results hard -- different markets and application areas, different business climate, and different past decisions affecting performance. A controlled, scientific study is impossible. Immature organizations rarely have the data to support good cost-benefit analyses. He concludes however that Process Improvement works: cost benefits of 6:1, 2X to 3X productivity improvement, 100X reduction in delivered defects.

The author points out that with increasing maturity the accuracy of cost and schedule estimates increase, with reduced variance in target date, and cost prediction improves. As organizations approach level 2, project level results represent the impact of a cluster of process changes. The first benefit is usually the ability to meet schedule.

"DACS Data Collection Surveys", DACS, July 2003.

In Issue 6-1 of the DoD Software Technical News, The DACS included an SPI Data Collection Survey so organizations could report the benefits from using SPI. The surveys included 12 attributes of product improvement, including defect reduction, productivity, and ROI. Two agencies responded: Goldman, Sachs, and Company of New York City, and HQ USAF in Washington, DC. Both organizations reported positive benefits in all areas applicable to their organizations.

Davis, N., and Mullaney, J., <u>The Team Software Process (TSP) in Practice: A Summary of recent Results</u>, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2003 (CMU/SEI-2003-TR-014).

This report provides results and implementation data from projects and individuals that have adopted the TSP. The results show that TSP teams can deliver essentially defect-free software on schedule while at the same time improving productivity. The report also illustrates adoption experiences of practitioners in the field, including TSP team members, their managers, and their coaches and instructors.

Diaz, M., Sligo, J., "How Software Process Improvement Helped Motorola," *IEEE Software*, Volume 14 #5 (September/October 1997), pp. 75-81.

This article demonstrates the data and metrics of the results of Motorola's CMM usage going from level 1 through level 5 (self assessments). Of particular note is the recognition by the authors that the most significant cost benefit occurs when projects finish early, allowing the company to apply more resources to the acquisition and development of new business. Productivity is directly related to our ability to win new programs in DoD and drives profitability in emerging commercial products.

Some observations of particular note include: They use Quality, Cycle Time & Productivity to evaluate their programs because this is what customers value. Each level increase of the CMM improves quality by 2X. Higher maturity projects have a better schedule performance index, however a decrease in cycle time between level 2 and 3 was surprising, but author

indicates the decrease from 2 to 3 may indicate a weak correlation between schedule performance and maturity that has been shown in other surveys. Defect injection rate is roughly 1/2 for each level, thus level 2 rework = 8X level 5 project. Productivity improves with increasing maturity level, but a decrease in productivity between level 2 and 3 appears to be a side effect of asking people to do too many things differently at level 3. They estimate a 677% ROI from their improvement efforts.

Diaz,M., King, J., "How CMM Impacts Quality, Productivity, Rework, and the Bottom Line", *Crosstalk*, Volume 15 #3 (March 2002), pp. 9-14.

This article explores various cost/benefit issues and examines performance results of various General Dynamics Decision Systems projects with relation to software process maturity. The quantitative data presented indicates CMM-based SPI yields dividends in terms of both higher productivity and higher software quality. Each level of improvement significantly cuts defect density, improves productivity, and reduces rework.

Dion, R., "Process Improvement and the Corporate Balance Sheet," *IEEE Software*, July 1993, pp. 28-35.

This paper summarizes the financial and non-financial impacts of Raytheon's Software Systems Laboratory (SSL) process improvement program, the Software Engineering Initiative, on Raytheon's balance sheet between 1988 and 1992. The author estimates that Raytheon's process improvement program resulted in a 7.7:1 return on investment, a two-fold increase in productivity and an SEI CMM rating increase from level 1 (Initial) to level 3 (Defined).

Raytheon's improvement program focused on policy and procedures, training, tools and methods, and a process database. Their process improvement paradigm involves a 3-phase process-stabilization, process-control and process-change process. Raytheon was surprised to see that benefits from this paradigm were observed during the first two phases, whereas they expected a full cycle would be necessary before an impact could be seen.

The author's analysis focuses on savings from less rework because of better inspection procedures. He estimates that they eliminated $9.2 million in rework costs. The author also estimates that Raytheon achieved a 130% productivity increase over this same period. Improved competitive position, higher employee morale, and lower absenteeism and attrition were second order effects of the improvement program.

Doolan, E. P., “Experience with Fagan's Inspection Method,” *Software Practice and Experience*, Vol. 22(2), February 1992, pp. 173-182.

This paper describes the use of Fagan’s inspection techniques at Shell Research’s Seismic Software Support Group. They used this technique for verifying and validating requirements. The author reports that 50% of all enhancement requests dealt with requirements issues that should have been found during requirements analysis. The author then describes their analysis of the price of this non-conformance.

The author describes the Fagan inspection process and the payback achieved from their inspection process. He states that fixing software in released software can be as much as 80X as expensive as fixes during the specification stage. The estimated ROI is 30:1.

Dorofee, A. J., Walker, J. A., Williams, R. C., "Risk Management in Practice," *Crosstalk*, Volume 10 #4 (April 1997), pp. 8-12.

This article summarizes lessons learned by the SEI over the last seven years in application of its' risk management program. The SEI's functions of managing risks includes: identify, analyze, plan, track, control, and communicate. Each organization involved in development is responsible to manage their own risks and everyone must work jointly to manage the risks to the program. The article also summarizes the common mistakes as well as transitioning and implementation advise of risk management.

Lessons learned include: (1) Written risks are harder to ignore than verbal concerns. A risk information sheet can be used to document risks; (2) Quantitative analysis is not always necessary, and Quantitative Analysis is only needed for risks that require numerical justification or rationale for mitigation planning; (3) Group related risks; (4) Prioritize and sort risks - not all risks can be mitigated; (5) Metrics and Measures - track both the risk and the mitigation plan. Spreadsheets that summarize all open risks are good for an overall view of the program's risks; and (6) Use databases to document risks, problems, schedules and change tracking, collect and analyze lessons learned.

Common mistakes include: (1) Risk management is not free. Allocate a % of project budget for mitigation costs; (2) How many risks were closed this week is the wrong question. Rather look at how many problems occurred that you did not foresee, then analyze why they were unforeseen; (3) Communication is important; and (4) don't just identify risks - do something with them.

Drobka, J., Noftz, D., Raghu, R., "Piloting XP on Four Mission-Critical Projects", IEEE Software, Volume 21 No. 6, November/December 2004, pp. 70 - 75.

This article asserts that software development teams constantly battle to increase productivity while maintaining or improving quality. It explains how four Motorola teams piloted Extreme Programming to see if it would let them satisfy their customers' constantly changing requirements while improving productivity.

El Emam, K., *The ROI From Software Quality*, Auerbach Publications, Boca Raton, FL, 2005.

This book provides the tools needed for software engineers and project managers to calculate how much they should invest in quality, what benefits the investment will reap, and just how quickly those benefits will be realized. It provides quantitative models necessary for making real and reasonable calculations and shows how to perform ROI analysis before and after implementing a quality program. The contents are supported with large amounts of data and numerous case studies.

Ezran, M., Morisio, M., and Tully, C., *Practical Software Reuse*, Springer-Verlag, London, UK, 2002.

This book seeks to emphasize the practice of reuse, and the practical issues that influence success or failure. It also seeks to offer a concise coverage of all the important aspects of reuse, focusing on the essentials of the subject rather than going into undue depth and detail on some topics at the expense of others.

Fagan, M. E., "Advances in Software Inspections," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 7, July 1986, pp. 744-751.

This paper describes benefits achieved from a formal inspection process. Inspections are formal processes that are intended to find defects in software nearer the point of injection of the defect than testing does, using less resources for rework. This is achieved by inspecting the output product(s) of each operation in the development process to verify that it satisfies the exit criteria of the operation. Defects are defined as any deviation from the exit criteria. Inspections can be performed on any product (e.g. test plans, procedures, users manuals) to improve defect detection efficiency of any process that creates a product.

The author, a member of the IBM technical staff, claims that the inspection process finds 60-90% of all defects and provides feedback to programmers that enables them to avoid

injecting defects in future design and coding work. The author claims that inspection costs typically amount to 15% of project cost. The article includes a revealing graph that shows the “snail” shaped graph of development resources vs. time without inspections and overlays on top of the same graph with inspections. The graph shows that resource requirements are slightly greater during planning, requirements definition and design. However the payoff occurs during coding and test when it is much more expensive to fix defects introduced earlier. The graph also shows that the overall development schedule is greatly reduced with inspections.

Fenton, N., “How Effective Are Software Engineering Methods?,” *Journal of Systems and Software*, Vol. 22, Number 2, August 1993, pp. 141-146.

This article is a skeptical view of the statistics and ROI data reported in the literature. The author claims that there is a poor state of the art of empirical assessment data in software engineering because of inappropriate or inadequate use of measurement. The article examines the quantitative benefits that 25 years of R&D in software engineering have brought. The author shows that anecdotal “evidence” of significantly improved quality and productivity are not backed up by hard empirical evidence. And where hard data exists, it is counter to the view of the so-called experts.

The author states that many of the best projects do not have state of the art methodologies or extensive automation and tooling. Rather they rely on strong teamwork, project communication and project controls. He believes that good management and organization is a more critical success factor than advanced technology.

Ferguson, P., Software Process Improvement Works! Advanced Information Services, Inc., Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1999 (CMU/SEI-99-TR-027).

Advanced Information Services (AIS) has seen considerable benefits from use of the Software Engineering Institute’s Capability Maturity Model (CMM) and Personal Software Process (PSP). AIS presents an overview of the organization software process improvement efforts and benefits in schedule, effort, quality, productivity, and employee satisfaction from use of CMM and PSP.

Ferguson, P., "Capability Maturity Model (CMM) + Personal Software Process (PSP) = Results", Software Engineering Process Group 2000, March 2000, Software Engineering Information Repository - Carnegie Mellon.

This presentation describes the results of using the CMM and PSP at Advanced Information Systems, Inc.

Freed, D., “CMMI Level 5: Return on Investment for Raytheon N TX”, Fourth Annual CMMI Technical Conference and User Group, Denver, CO, November, 2004.

This presentation describes activities at the Software Engineer division of StorageTek, a large company that designs, manufactures, sells, and maintains data storage hardware and software. The SPI effort goals were mapped to corporate quality, cost, and delivery goals. An interim goal of achieving Software Engineering Institute (SEI) Capability Maturity Model (CMM) Level 2 was set. Eventually, Capability Maturity Model â€“ Integrated key process areas were satisfied with a process combining and Agile Development. Organizational transformation techniques were applied to gain acceptance of the newly designed process. The presentation documents benefits in productivity gains and less schedule slippage.

Freimut, B., Briand, L., and Vollei, F., “Determining Inspection Cost-Effectiveness by Combining project Data and Expert Opinion”, *IEEE Transactions on Software Engineering*, Vol. 31, No. 12 (December, 2005), pp. 1074-1092.

There is a general agreement among software engineering practitioners that software inspections are an important technique to achieve high software quality at a reasonable cost. However, there are many ways to perform such inspections and many factors that affect their cost-effectiveness. It is therefore important to be able to estimate this cost-effectiveness in order to monitor it, improve it, and convince developers and management that the technology and related investments are worth while. This work proposes a rigorous but practical way to do so. In particular, a meaningful model to measure cost-effectiveness is proposed and a method to determine cost-effectiveness by combining project data and expert opinion is described. To demonstrate the feasibility of the proposed approach, the results of a large-scale industrial case study are presented and an initial validation is performed.

Galorath, D., “Software Reuse and Commercial Off-the-Shelf Software”, *IT Metrics and Productivity Journal*, August 28, 2007, pp. 1-22.

Organizations faced with the difficulties and costs associated with the development of software have turned to the reuse of existing software or using commercial off-the-shelf (COTS) software as an option. Reuse, whether involving home-grown or COTS components, certainly promises lower cost, better quality, a decrease in risk, and the potential for a less stressful development process. Many such efforts succeed, but the promises of decreased cost and risk are not always realized. Requirements, algorithms, functions, business rules, architecture, source code, test cases, input data, and scripts can all be reused.

Galin, D. and M. Avrahami (2005). “Do SQA Programs Work – CMM Works. A Meta Analysis”, *Proceedings of the International Conference on Software – Science, Technology & Engineering (SwSTE’05)* (Ed. by A. Tomer and S. R. Schach), Washington, DC, Computer Science Press: 95-100.

Many software development professionals and managers of software development organizations are not fully convinced in the profitability of investments for the advancement of software quality assurance (SQA) systems. The results included in each of the articles cannot lead to general conclusions on the impact of investments in upgrading an SQA system. The meta analysis was based on CMM level transition (CMMLT) analysis of available publications and was for the seven most common performance metric. The CMMLT analysis is applicable for combined analysis of empirical data from many sources. Each record in the meta analysis database is calculated as “after-before ratio”, which is nearly free of the studied organization’s characteristics. Because the CMM guidelines and SQA requirement are similar, it is claimed that the results for CMM programs are also applicable to investments in SQA systems The extensive database of over 1,800 projects from a variety of 19 information sources leading to the meta analysis results – proved that investments in CMM programs and similarly in SQA systems contribute to software development performance.

Gibson, D., Goldenson, D., and Kost, K., Performance Results of CMMI-Based Process Improvement, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006 (CMU/SEI-TR-2006-004).

This SEI technical report summarizes much of the publicly available empirical evidence about the performance results that can occur as a result of of CMMI-based process improvement. The report contains a series of 10 brief case descriptions that were created in

collaboration from representatives of several organizations that have realized notable quantitative performance results through their CMMI-based process improvement efforts.

Goldenson, D., and Gibson, D., Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006 (CMU/SEI-2003-SR-009).

There is a widespread demand for evidence about the impact and benefits of process improvement based on Capability Maturity ModelÂ® Integration (CMMIÂ®) models. Much has been documented about the practice of CMMÂ®-based process improvement and its value for the development and maintenance of software and software-intensive systems; however, the existing information is sometimes outdated and there are increasing calls for evidence directly based on CMMI experience. This special report presents selected results from 12 case studies drawn from 11 organizations. While still limited, the case studies provide credible evidence that CMMI-based process improvement can help organizations achieve better project performance and produce higher quality products. The report also describes plans for gathering further evidence from organizations using CMMI models.

Goyal, A., Kanungo, S., Muthu, V., Jayadevan, S., "ROI for SPI: Lessons from Initiatives at IBM Global Services India", Third Software Engineering Process Group in Asia Conference, January 2001.

One of the most critical outcomes of software process improvement (SPI) efforts is the realization of return on investment (ROI). However, the notion of ROI, in the context of SPI, is associated with two problem areas. One has to do with conceptualizing and defining what can be loosely referred to as cost and benefits. The other problem is to come up with an integrative framework to understand how seemingly disjointed costs and benefits relate to each other. While these two sets of problems present from a methodological standpoint â€“ and, as a consequence, lend themselves to methodological resolutions, we posit a stance that goes beyond numbers. In doing so, we are suggesting a conceptualization of ROI that derives from shared mental models whereby SPI techniques become deeply ingrained into individuals belief systems. In that context, the question does not remain “”what is ROI”? The question of relevance evolves into “Can we address the appropriateness of ROI”? In essence we have written this paper because, in general, individuals have difficulty in (or problems with) figuring out how to calculate ROI using simple information and their management is always

concerned about this question. This paper addresses three things: (i) ways to calculate ROI, (ii) middle managers' buy in, and (iii) overall conceptualization of ROI in SPI.

Harrison, W., Raffo, D., Settle, J., and Eickelmann, N., "Technology Review: Adapting Financial Measures: Making a Business Case for Software Process Improvement", Software Quality Journal, V. 8, N. 3 (Nov 1999): 211-231.

Software firms invest in process improvements in order to benefit from decreased costs and/or increased productivity sometime in the future. Such efforts are seldom cheap, and they typically require making a business case in order to obtain funding. The authors review some of the main techniques from financial theory for evaluating the risk and returns associated with proposed investments and apply them to process improvement programs for software development. The authors also discuss significant theoretical considerations as well as robustness and correctness issues associated with applying each of the techniques to software development and process improvement activities. Finally the authors introduce a present value technique that incorporates both risk and return that has many applications to software development activities and is recommended for use in a software process improvement context.

Harter, D., Krishnan, M., and Slaughter, S., "Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development", Management Science, April 2000, pp. 451 - 466.

This paper investigates the relationship between process maturity, quality, cycle time, and effort for the development of thirty software projects by a major information technology (IT) firm. Higher levels of process maturity as assessed by the Software Engineering Institute's Capability Maturity Model are associated with higher product quality, but also with increases in cycle time and development effort. However, the findings indicate that the reductions in cycle time and effort due to improved quality outweigh the increases from achieving higher levels of process maturity. Therefore, the net effect of process maturity is reduced cycle time and development effort.

Hausler, P. A., Linger, R. C., Trammell, C. J., "Adopting Cleanroom Software Engineering With A Phased Approach," *IBM Systems Journal*, Vol. 33, No. 1, 1994, pp. 89-109.

This paper describes key Cleanroom technologies and summarizes quality results achieved by Cleanroom teams. It also describes a three phase approach to Cleanroom implementation

based on the software maturity level of an organization and summarizes the results of a large IBM Cleanroom project that successfully applied a phased approach.

Cleanroom software engineering is a managerial and technical process for the development of software approaching zero defects with certified reliability. It provides a complete discipline within which software teams can plan, specify, design, verify, code, test and certify software. In this approach, the more powerful process of team correctness verification replaces unit testing and debugging, and software enters system testing with no execution by development teams. All errors are accounted for from the first execution, with no private unit testing permitted. Certification test teams are not responsible for testing-in quality, but rather for certifying the quality of software with respect to its specification. Cleanroom software typically enters system test approaching zero defects. Successive increments elaborate the top down design of increments already in execution.

Hayes, W. (1999). “Research Synthesis in Software Engineering: A Case for Meta-Analysis”, *Proceedings of the Sixth International Software Metrics Symposium*: 143-151.

The use of meta-analytic techniques to summarize empirical software engineering research results is illustrated using a set of 5 published experiments from the literature. The intent of the analysis is to guide future work in this area through objective summarization of the literature to date. A focus on effect magnitude, in addition to statistical significance is championed, and the reader is provided with an illustration of simple methods for computing effect magnitudes.

Hayes, W., Zubrow, D., “Moving On Up: Data and Experience Doing CMM-Based Software Process Improvement,” Presentation at the *Seventh Software Engineering Process Group Conference,* Boston, MA, May 23, 1995.

In this presentation, the authors provide information about organizations that have gone through reassessments, the growth of reassessments spanning 1987-1994, the types of organizations that have had reassessments and the relative maturity profile of these organizations.

The presenters conclude that to move from Level 1 CMM to Level 2 requires, on average, 30 months, and from Level 2 to Level 3, on average, 25 months. They identify issues that most clearly distinguish Level 2 organizations from Level 1 organizations.

Henry, D., McCarthy, L., and Chitnis, S., "CMMI Transition at Motorola GSG", CMMI Technology Conference, Denver, CO, Nov 2003.

This presentation describes the improvements made at the Motorola Global Software Group (GSG) India location from 2001 to 2003 as they attained CMMI Level 5. The organization met most of its improvement goals in areas such as customer satisfaction, cycle time reduction, post release defects, and productivity. The presentation also describes current efforts at the GSG Canada location to improve process maturity.

Herbsleb, J., Zubrow, D., Siegel, J., Rozum, J., Carleton, A., "Software Process Improvement: State of the Payoff," *American Programmer*, Vol. 7 no. 9, September 1994, pp. 2-12.

This paper provides statistical results as reported by 13 organizations (companies, DoD organizations) to show what benefit or value can be gained by organizations involved in serious CMM-based software process improvement (SPI). The article then points out that beyond the basic Return on Investment (ROI) numbers, we need to understand those factors in the SPI process that increase successes and those that result in a failure.

Results reported include costs/software engineer/year for SPI, percentage gain/year in productivity (lines of code/year), percentage reduction/year in calendar time to develop software, percentage reduction/year in post-release defects and ROI for SPI efforts.

Definitions of note: ROI - ratio of *measured benefits* to *measured costs*. *Measured benefits* typically include savings from productivity gains and fewer defects. *Measured costs* of SPI generally include the costs of the Software Engineering Process Group (SEPG), assessments and training. *Measured costs* do not include staff time to put new processes in place.

Hodgetts, P., and Phillips, D., "Extreme Adoption Experiences of a B2B Start-up", Extreme Programming Perspectives, 2003, Addison-Wesley, pp. 355 - 362.

This book chapter presents the results of adopting XP at an Internet business-to-business (B2B) start-up. The writers discuss their motivations and goals for adopting XP and the context under which the adoption efforts were evaluated. Then they present the results in terms of objective metrics and subjective evaluations. The results indicate that the project conducted under XP demonstrated significant improvements.

Hoffman, Gabriel, "Integrating PSP and CMMI Level 5, Northrop Grumman", 3rd Annual CMMI Technology Conference and User Group, May, 1 2003, pp. 26 - 34.

This briefing describes how Northrop-Grumman combined use of the Personal Software process (PSP) and CMMI Level 5 to enhance the ROI and reduce defects for an Air Force program they were managing. Significant improvements were noted in both areas.

Huang, L., and Boehm, B., "How Much Software Quality Investment Is Enough: A Value-Based Approach". *IEEE Software*, Volume 23, Issue 5 (September-October 2006), pp. 88-95.

A classical problem facing many software projects is determining when to stop testing and release the product for use. Risk analysis helps address this issue by balancing the risk exposure of doing too little with the risk exposure of doing too much. A quantitative approach based on the COCOMO II cost-estimation model and the COQUALMO quality-estimation model helps answer the question, "How much software quality investment is enough?" The authors use the models and some representative empirical data to assess the relative payoff of value-based testing as compared to value-neutral testing. They include examples of the approach's use under differing value profiles.

Humphrey, W.S., *Managing the Software Process*, Addison-Wesley, Boston, MA, 1989.

Watts S. Humphrey, drawing on years of experience at IBM and the SEI, provides here practical guidance for improving the software development and maintenance process. He focuses on understanding and managing the software process because this is where he feels organizations now encounter the most serious problems, and where he feels there is the best opportunity for significant improvement. Both program managers and practicing programmers, whether working on small programs or large-scale projects, will learn how good their own software process is, how they can make their process better, and where they need to begin

Humphrey, W. S., Snyder, T. R., Willis, R. H., "Software Process Improvement at Hughes Aircraft," 1991, *IEEE Software*, 8 (4), pp. 11-23

This article describes ROI information from software process improvement efforts at the Software Engineering Division of Hughes Aircraft. Areas addressed by Hughes are discussed, along with costs associated with those improvements. The authors describe the yearly savings achieved. Certain non-quantifiable benefits of the process, such as lower software professional turnover are also discussed.

Humphrey, W.S., The Personal Software Process, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000 (CMU/SEI-TR-2000-022).

Humphrey, W.S., The Team Software Process, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000 (CMU/SEI-TR-2000-023).

These are companion documents which describe the Personal Software process (PSP) and Team Software Process (TSP) respectively, and can be looked upon as source documents for these processes. Both the PSP and TSP are offshoots of the SEI Capability Maturity Model (CMM) developed in the late 1980s.

The PSP extends the CMM to the people who actually do the work; it concentrates on the work practices of individual engineers. The principle behind the PSP is that, in order to produce quality software systems, every engineer who works on the systems must do quality work.

While the PSP focuses on individual skills, the TSP emphasizes the need for people to work together as cohesive units. It is based on the principle that engineering teams can do extraordinary work, but only when they are properly formed. One feature of the TSP is the “team launch” process where teams participate in nine meetings in four days to develop a detailed plan and be committed to the plan the team develops.

Humphrey, W.S., "Being a Software Professional", Software Engineering Process Group 2004, March 2004, pp. 18 – 19.

This presentation describes the results of companies improving CMM then using the TSP when they achieved CMM Level 5. Estimating accuracy and defect reductions occurred at higher CMM levels and when the TSP was used.

Jarzombek, J., "Enterprise-Wide Perspective for Managing Process Improvement", Twelfth Annual Software Technology Conference, Salt Lake City Utah, 30 April - 5 May 2000.

An enterprise-wide perspective for managing process improvement can allow a large corporation to provide corporate sponsorship, guidance, and a multi-discipline perspective to software process improvements. A business strategy can be integrated in the program as well as organization process relationships and a process maturity plan. This presentation looks at an enterprise-wide process improvement program called the Computer Resources Support Improvement Program (CRSIP) at the Ogden Air Logistics Center located at the Hill Air Force Base in Utah.

Jensen, R., “A Pair Programming Experience”, Crosstalk, Volume 16 #3 (March, 2003), pp. 22-24.

Agile software development methods, including extreme programming, have risen to the forefront of software management and development interest during the last few years. The "Agile Manifesto" published in 2001 created a new wave of interest in the agile philosophy and re-emphasized the importance of people, along with the idea of "pair programming." As defined, pair programming is two programmers working together, side by side, at one computer collaborating on the same analysis, design, implementation, and test. The author was introduced to teamwork and pair programming indirectly as an undergraduate electrical engineering student in the 1950s. Later in 1975, he was asked to improve programmer productivity in a large software organization. The undergraduate experience led him to an experiment in pair programming. The very positive results of this experiment are the subject of the case study in this article.

Jensen, R., “An Economic Analysis of Software Reuse”, Crosstalk, Volume 17 #12 (December, 2004), pp. 4-8

This article presents a simplified economic analysis of the cost of software reuse. The reuse definition used here includes both commercial off-the-shelf (COTS) and existing software from an upgraded platform. The results are independent of software estimating tools or models. The model used in this analysis relates the cost of software development to the reused software level and the costs of developing and maintaining the software components. COTS software is a special case of reuse described in this article.

Jones, C., “Software Defect Removal Efficiency”, *Computer*, April 1996, Vol. 29, No. 4, pp. 94-95.

The author describes various aspects of software defect removal efficiency as it applies to the US software industry. Software defect removal efficiency is the percentage of total bugs eliminated before the release of a product. The author views this as a good metric for choosing defect removal operations that maximize efficiency and minimize cost and schedule.

The article describes how the top companies achieve a greater than 95% software defect removal efficiency. The author claims that high levels of customer satisfaction correlate strongly with high levels of defect removal efficiency.

Jones, C., Software Assessments, Benchmarks, and Best Practices, Addison-Wesley, Boston, MA, 2000.

In this book, the author explains qualitative and quantitative approaches to software development analysis in three areas, assessments, benchmarks, and process improvements. Assessments and benchmark studies can show an organization its current strengths and weaknesses, and where to emphasize process improvements. The author presents a 7-stage process improvement program of which the first stage consists of assessments and benchmarking. And the other six are actual process improvement activities. The author also shows how to assess cost, timing, and value of process improvement activities for specific programs.

Jones, C., Estimating Software Costs: Bringing Realism to Estimating (Second Edition), McGraw-Hill, New York, 2007.

This book is written to provide a clear, complete understanding of how to estimate software costs, schedules, and quality using real-world information contained in this book. The book discusses how to choose the correct hardware and software tools, develop an appraisal strategy, deploy tests and prototypes, and produce accurate software cost estimates.

This fully updated and expanded volume provides cost-estimating methods for projects using modern technologies including Java, object-oriented methods, and reusable components. Written by a pioneer and leading authority in the field of software estimation, this new edition is the definitive resource for developers building complex software.

Jones, L., "Product Line Acquisition in DoD: The Promises, the Challenges" Crosstalk, Volume 12 #8 (August 1999), pp. 17-21.

Industrial use of software product line technology has resulted in some impressive savings while also improving product quality and delivery time. Although there has been some successful use of this technology within the Department of Defense (DoD), there are special challenges. This article reports the results of two DoD product line workshops in which important issues and successful practices were shared.

Joos, R., “Software Reuse at Motorola,” *IEEE Software*, September, 1994, pp. 42-47.

The author summarizes the three phase approach followed by Motorola to achieve effective reuse. The three phases include the grass roots beginning, software management involvement and use of tools and technology.

The article describes the approach taken on two pilot projects and the results achieved in these pilot. A cash reward incentive program facilitated reuse. An 85% reuse rate and a 10:1 productivity savings was achieved. The article concludes with recommendations by the author to others that want to initiate a reuse program.

Kelly, J. C., Sherif, J. S., Hops, J., "An Analysis of Defect Densities Found During Software Inspections," *Journal of Systems Software*, 1992; Vol. 17, pp. 111-117

This article describes an analysis of factors influencing the defect density of products undergoing software inspection at the Jet Propulsion Laboratory that require a high level of quality. Inspections detect errors as early as possible in the development lifecycle. The authors describe the steps involved in performing inspections, which have been tailored from Fagan inspections (Fagan 86).

JPL tailored Fagan to improve the quality of software requirements, architectural design, detail design, source code, test plans, and test procedures. Also JPL added to Fagan a "third hour" step which includes time for team members to discuss problem solution and to clear up open issues raised in inspections.

The results from 203 inspections are summarized. The authors develop a model of defects found based on the phase of development being inspected. The average cost to fix defects in early phases, versus later phases, is also described. The authors provide some guidelines for conducting reviews.

Kimberland, K., "Microsoft's Pilot of TSP Yields Dramatic Results", Carnegie Mellon News @ SEI No. 2, 2004, pp. 4 - 6.

This article summarizes the results of Microsoft's use of the Team Software Process (TSP) to unite a group of frustrated individual programmers into a team. The results were a reduction in defects and a cost savings.

Kossiakoff, A., and Sweet, W., *Systems Engineering: Principles and Practices*, John Wiley and Sons, Hoboken, NJ, 2003.

Systems Engineering Principles and Practice is designed to help readers learn to think like systems engineers, to integrate user needs, technological opportunities, financial and schedule constraints, and the capabilities and ambitions of the engineering specialists who have to build the system. The book devotes particular attention to knowledge, skills, mindset, and leadership qualities needed to be successful professionals in the field. This book is an outgrowth of the Johns Hopkins University Master of Science Program in Engineering, developed to meet an urgent and expanding need for skilled systems engineering in industry and government. The authors, who have sixty years of collective experience in this field, were part of the curriculum design team as well as members of the initial faculty. The book is used to support four core courses in the curriculum, and has been exhaustively classroom tested.

Krasner, H., "Accumulating the body of Evidence for The Payoff of Software Process Improvement", Krasner Consulting, November 19, 1997, pp. 5 - 13.

This paper addresses payoffs associated with SPI. Section 4, “Case Histories of SPI Payoffs”, documents several case histories of successful SPI implementation, including results from NASA, IBM, Hewlett-Packard, Raytheon, and Motorola – India. The organizations had SPI initiatives lasting from 5-10 years. The payoffs sometimes took several years to be observable through measurements, but were always noteworthy.

Layman, L., Williams, L., and Cunningham, L., "Exploring Extreme programming in Context: An Industrial Case Study”, *Proceedings of the Agile development Conference (ADC’04)*, IEEE, 2004.

A longitudinal case study evaluating the effects of adopting the Extreme Programming (XP) methodology was performed at Sabre Airlines Solutions. The case study compares two releases of the same product. One release was completed prior to the team’s adoption of the XP methodology, and the other was completed after two years of XP use. There were marked improvements in productivity and quality, which suggests that use of XP can result in improvements in these areas.

Leach, R., Software Reuse, New York, McGraw-Hill, 1997.

While this book will provide a complete description of software reuse, it focuses on methods for reuse that are feasible without major investments in new software methodology, as well as on cost estimation issues and on certification of reusable software components.

Lee, E., "Software Inspections: How to Diagnose Problems and Improve the Odds of Organizational Acceptance," _Crosstalk,_ Volume 10 #8 (August 1997), pp. 10-13.

This paper documents lessons learned by the author on Lockheed Martin's space shuttle onboard software project. On this project, formal inspections form the cornerstone of their quality program. On their project, they are able to achieve error detection rates of 85 to 90%. Within other projects, success is not as uniform. The author observes that inadequate training is one of the major causes of failures in inspections, especially inadequate training by moderators. This article is of importance to this paper because it continues to demonstrate successes with use of inspections.

Lierni, P., "The Affect of Agile Software Development Methods on Department of Defense (DoD) Systems Acquisition Programs", Technical Paper from Deputy Under Secretary of Defense for Acquisition and Technology (DUSD A&T) Software Engineering and Systems Assurance (SSA) Deputy Directorate, 24 January 2007

This paper describes the results of a 2006 survey to determine the current state of agile software development. More than 80% of the more-than 700 responding organizations do use agile methods. More than 25% of the organizations realized increased productivity, reduced cost and schedule, and reduced numbers of defects when they used agile methods. The paper does provide a large number of references on agile methods.

Lim, W. C., "Effects of Reuse on Quality, Productivity and Economics," *IEEE Software*, September, 1994, pp. 23-31.

Hewlett-Packard (HP) has found that reuse can have a significant and positive impact on software development. The article presents metrics from two HP reuse programs that resulted in improved quality, increased productivity, shortened time-to-market and enhanced economics resulting from reuse. The information presented summarizes findings at two HP facilities.

Statistics presented include reuse percentages, defect reduction and productivity improvements from reuse. Costs to create reusable components are also presented. The effort increase by phase to create reusable components is discussed.

Linger, R. C., "Cleanroom Software Engineering for Zero-Defect Software," *Fifteenth International Conference on Software Engineering*, 1993, pp. 2-13.

This article describes characteristics and benefits of Cleanroom software development. Cleanroom software engineering teams are developing software with zero defects with high probability and with high productivity. Correctness is built in by the development team through formal specification, design and verification. Team correctness verification replaces unit testing and debugging, and software enters system testing directly, with no execution by the development team. All errors are accounted for from first execution. The certification team does not test in quality, but rather certifies the quality of software with respect to its design. Cleanroom development is being successfully used at IBM and other organizations.

The author cites reliability figures from 15 software projects. Product development schedule comparisons are made and the author describes the box structure of specifications (black, state and clear).

Lipke, W., Butler, K., "Software Process Improvement: A Success Story," *CrossTalk*, Number 38, November 1992, pp. 29-31, 39.

This article provides an overview of the Aircraft Software Division (LAS) of the Oklahoma City Air Logistics Center (OC-ALC), Tinker AFB, Oklahoma, process improvement efforts, assessments, and lessons learned. LAS has a technical and management team to oversee their process improvements. They feel these teams are the single most important key to the success of process improvement.

Their SEI evaluation identified 44 improvements. ROI data on 18 projects is summarized. Intangible improvements included increased communications, increased customer satisfaction and on time and within budget software delivery.

Lougee, Hoyt, "DO-178B Certified Software: A Formal Reuse Analysis Approach", *CrossTalk*, Volume 18, #1 (January, 2005), pp. 20-25.

This article discusses software reuse as an alternative to designing from scratch for a next-generation system. Reuse can provide significant return on investment and time-to-market advantages; however, one must be rigorous in his or her approach to planning, analyzing, executing, and tracking reuse. While this article highlights DO-178B certified software, the ideas presented can be applied to almost all software programs.

Madachy, R., "Process Improvement Analysis of a Corporate Inspection Program," *Seventh Software Engineering Process Group Conference,* Boston, MA, May 23, 1995.

This paper discusses return on investment and defect prevention results from the Litton Data Systems inspection process. Over 400 people were trained and 600 inspections were performed utilizing this process. The inspection process is similar to Fagan's (1986) inspection process, but Litton has made several modifications.

Litton has experienced a 30% reduction of errors found during systems integration and system test. Their division had set goals of saving at least 50% of integration effort by spending more effort during design and coding for inspections. They have achieved this objective in one major project. Other results reported include 2.3 person hours saved in testing for every inspection hour; 73% of all 600 inspections have produced a positive return; and 3% of the total project effort was used for inspections.

Manzo, John, "Odyssey and Other Code Science Success Stories", *CrossTalk*, Volume 15, #10 (October, 2002)), pp. 19-21, 30.

This article describes the success achieved using Code Science, an agile software development method based on eXtreme Programming (XP), to develop a complex industrial automation application. With a brief review of XP as background, code science is described in terms of refinements made to XP in applying it to a wide variety of application domains and industries over a period of almost four years. Included are real-world insights from the developers' experience in applying this agile development method, concluding with a quantitative measure of the effectiveness of XP since its inception almost four years ago.

Matsumura, K., "Software Reuse - What is Different With Ordinary Software Development", *Proceedings of the Thirteenth International Conference on Software Engineering*, IEEE, Austin, Texas, 13-16 May 1991, pp. 55-57.

This article was a part of a panel discussion on software reuse. The author discusses the application of a standard component approach to reuse used at Toshiba for several projects. The application of reuse resulted in a defect reduction of 20% to 30% during integration testing.

Maurer, F., and Martel, S., "Extreme Programming: Rapid Development for Web-Based Applications", *IEEE Internet Computing*, Volume 6, Issue 1 (January-February 2002), pp. 86-90.

Agile processes like extreme programming (XP), Scrum, Crystal, and adaptive software development aim to increase a software organization's responsiveness while decreasing

development overhead. They focus on delivering executable code and see people as the strongest ingredient of software development. We offer an overview of the philosophy and practice behind XP, which is currently the most popular agile methodology.

McAndrews, D., The Team Software Process: An Overview and Preliminary Results of Using Disciplined Practices, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000 (CMU/SEI-TR-2000-015).

This report describes the TSP technology as an implementation strategy for teams that are attempting to apply disciplined software process methods. It provides some of the background and rationale for the TSP approach, as well as an overview of the technology. Then, the report presents initial results of the use of the TSP technology in four different organizational settings. In each of these organizations, the data show that defect densities found in system- level test activities and the actual duration of these system-level tests were reduced significantly with the use of the TSP. In addition, the accuracy of software estimates improved, and the variation in estimation accuracy was significantly reduced. Based on the analysis of these results, some assertions are made to help organizations set goals for improvement.

McCann, B., "When Is It Cost-Effective to Use Formal Software Inspections", *Crosstalk*, Volume 17 #3 (March, 2004), On-Line Article.

This article presents a method quantitatively to determine the parametric limits to cost-effectiveness of software code inspections. The analysis presented leads to the conclusion that it is cost-effective to inspect both initial code and modifications made to the code after initial coding. Any exceptions should be carefully considered based on quantitative analysis of the projected impact of the exceptions. Also, any proposed substitution for rigorous inspections should be carefully evaluated for cost-effectiveness prior to replacing or modifying the process.

McGarry, F., Jeletic, K., "Process Improvement as an Investment: Measuring Its Worth," NASA Goddard Space Flight Center, Software Engineering Laboratory, SEL-93-003, 1993.

This paper discusses process improvement and measuring, from a Return on Investment perspective, the benefits achieved from the improvement. The article compares and contrasts the SEI CMM to the NASA SEL improvement model and how improvements are measured

in each model. The article then details the improvements observed at the SEL over an 18 year period, including improvements in reuse, development costs, and quality.

McGibbon, Thomas, "An Analysis of Two Formal Methods: VDM and Z", DoD Data Analysis Center for Software (DACS), Rome, NY, 20 August 1997.

This paper compares and contrasts the strengths and weaknesses of the Vienna Development Method (VDM) and Z in the software design life cycle phase, and compares and contrasts VDM and Z to other formal models. Tool support, lessons learned, and technical and achieved business benefits are emphasized. Based on available data, this paper analyzes the return-on-investment (ROI) from use of these methods, and this ROI data is compared to ROI data from cleanroom software engineering and other process improvement methods.

Miller, J. (2000). "Applying Meta-Analytical Procedures to Software Engineering Experiments", *Journal of Systems and Software*, V. 54, N. 1: 29-39.

The treatise of this paper is: Can meta-analysis be successfully applied to current software engineering experiments? The question is investigated by examining a series of experiments, which themselves investigate which defect-detection technique is best. Applying meta-analysis techniques to the software engineering data is relatively straightforward, but unfortunately the results are highly unstable. The meta-analysis shows that the results are highly disparate and do not lead to a single reliable conclusion. The reason for this deficiency is the excessive variation within various components of the experiments. The paper outlines various ideas from other disciplines for controlling this variation and describes a number of recommendations for controlling and reporting empirical work to advance the discipline towards a position, where meta-analysis can be profitably employed.

Millot, Philippe, "Thomson CSF - SPI Return on Investment (ROI) Calculation - Lessons Learned", European Software Engineering Process Group 1998, 1999, Software Engineering Information Repository - Carnegie Mellon, pp. 1 - 14.

This presentation describes the return on investment realized by Thomson-CSF as they advanced to CMM Level 2 and, later, to CMM Level 3. Not only did ROI improve, but defects were also reduced.

Mills, H.D., Dyer, M., Linger, R.C., "Cleanroom Software Engineering," *IEEE Software*, September 1987, pp. 19-24.

This article, written by the developer of Cleanroom software engineering, Harlan Mills, describes early successes employing this methodology. The authors believe that software can be engineered under statistical quality control and that certified reliability statistics can be provided with delivered software.

Cleanroom development is an incremental development methodology and the authors describe the typical size of increments. The productivity and schedule impact of Cleanroom development is also discussed.

Munson, R., "How the TSP Impacts the Top Line", *Crosstalk*, Volume 15 #9 (September, 2002), pp. 9-11.

The thrust of this article is to show how productivity improvements from practices such as the TSP affect a corporation's profitability, or "top line". This article compares the development costs associated with teams in a traditional test-based organization to TSP teams. The article also presents product and quality data from several TSP projects at one industry organization.

Nawrocki, J. and Wojciechowski, A., "Experimental Evaluation of Pair Programming", Proceedings of ESCOM 2001, 2001, pp. 269 - 276.

Pair programming is a kind of collaborative programming where two people are working simultaneously on the same programming task. It is one of the key practices of eXtreme Programming. In the paper pair programming is compared with two variants of individual programming: one of them is based on Personal Software Process that has been proposed by W. Humphrey, and the other is a variant of eXtreme Programming tailored to individuals. Four experiments are described that has been performed at the Poznan University of Technology. During those experiments 21 students wrote 4 C/C++ programs ranging from 150 to 400 LOC. The obtained results are compared with the results of similar experiments described by J.T. Nosek and L. Williams, et al.

Northrop, L., "Software Product Lines: Reuse that Makes Business Sense", ASWEC 2006 Conference, 2007.

This presentation explains what software product lines are, the benefits of having them, and how to develop and manage product lines. Several examples of company use and associated benefits are given. The presenter shows how software product lines and associated systematic reuse are the hallmark of reuse in this decade as components were in the 1990s.

Nosek, J. T., "The Case for Collaborative Programming", Communications of the ACM, Volume 41 No. 3, March 1998, pp. 105 - 108.

Collaborative programming is a term used to refer to two or more programmers that are working together on the same code and algorithm. In an experiment that compares the performance of individual programmers vis-a-vis programmers working in pairs, it was observed that the teams created more functional and readable programs. In addition, levels of enjoyment and confidence are higher and less time is utilized to develop the programs.

O'Connor, J., Mansour, C., Turner-Harris, J., Campbell, G., "Reuse in Command-and-Control Systems," *IEEE Software*, September, 1994, pp. 70-79.

This paper discusses the authors experience at Rockwell International's C2 Systems Division (CCSD) with Software Productivity Consortium's Synthesis methodology for reuse Their experience has resulted in a partially automated environment that supports specification of systems and the generation of requirements, design and code.

The authors summarize the technology, the benefits derived from the technology, and the costs to create the environment. The payoff of the technology is also described.

Oldham, L.G., Putman, D.B., Peterson, M., Rudd, B., Tjoland, K., "Benefits Realized from Climbing the CMM Ladder", Crosstalk, Volume 12 No. 5, May 1999, pp. 7 - 10.

Industrial use of software product line technology has resulted in some impressive savings while also improving product quality and delivery time. Although there has been some successful use of this technology within the Department of Defense (DoD), there are special challenges. This article reports the results of two DoD product line workshops in which important issues and successful practices were shared.

Olson, T., "Piloting Software Inspections to Demonstrate Early ROI," Notes from Presentation given at the 1995 SEPG Conference

This presentation describes a software inspection pilot conducted at Cardiac Pacemakers, Inc. (CPI). CPI makes pacemakers and defibrillators that require life-critical software. Inspections are being used to improve the reliability of the software.

The paper describes some industry success stories with inspections, the inspection process and key ROI questions. The author compares the effort to fix a defect early in the lifecycle to the effort at the end of the development process.

O'Neill, D., "Determining Return on Investment Using Software Inspections" *Crosstalk,* Volume 16 #3 (March 2003), pp. 16-21.

This article examines the defined measurements used to form a derived metric for return on investment. These measurements involve additional cost multiplier, defect detection rate, cost to repair, and detection cost. The article further examines the behavior of these measurements and metrics for various software product-engineering styles using data collected from the National Software Quality Experiment.

Paulk, Mark C., Chrissis, Mary Beth, "Mastek Limited, Mumbai, India", The 2001 High Maturity Workshop, January 2002, pp. 57 - 63.

In March of 2001, the Software Engineering Institute (SEI) in Pittsburgh, PA, hosted a workshop for high maturity organizations to better understand practices that characterize Capability Maturity Model for Software (Software CMM) Level 4 and 5 organizations. Topics of discussion included practices described in the Software CMM as well as other practices that have a significant impact in mature organizations. Important themes included statistical process control for software, the reliability of Level 4 and 5 assessments, and the impact of the CMM Integration (SM) effort. Additional topics solicited from the participants included measurement, Six Sigma, Internet speed and process agility, and people and cultural issues. This report contains overviews of more than 30 high maturity organizations and the various working group reports from the workshop.

Phillips, M., "CMMI V1.2: What Has Changed and Why", *Crosstalk*, Volume 20 #2 (February 2007), pp. 4-7

Since the source document for each CMMI release is more than 500 pages in length, it is useful to have articles like this that summarize the current version of the CMMI and notable changes from previous versions. This article explains the major changes for Version 1.2 (released in 2006) and the rationale for these changes. Phillips is a most credible source since he is the Program Manager for CMMI Version 1.2 at the SEI.

This article, like most all others about the CMMI, assumes the reader has familiarity with the CMMI. Anyone who wants to use the CMMI or needs to learn about it should become familiar with the source document for the CMMI (referenced in this bibliography).

Pipp, W., "Software Process Improvement Pays", Twelfth Annual Software Technology Conference, Salt Lake City Utah, 30 April - 5 May 2000.

This presentation responds to the opinion that, with tight schedules and budgets, a company just can not afford software process improvement. Raytheon Missile Systems determined that pursuing CMM Level 5 would be worth the investment. They experienced positive returns when they advanced from CMM Level 2 to Level 4 from 1995 to 1998.

Pitterman, B., "Telcordia Technologies: The Journey to High Maturity", IEEE Software, Volume 17 No. 4, July/August 2000, pp. 89 - 96.

In early 1994, the software development groups across Telcordia Technologies underwent serious self-examination. The main quality indicators showed the business was in trouble. Field fault density was at 48 faults per 1,000 function points, well above the industry average. Customer satisfaction was at the 60% level. Telcordia had just failed the International Standards Organization (ISO) 9001 registration audit. In the past the organization had tried several times to improve product quality and customer satisfaction, but the results tended to be over-engineered. In 1994, a small team of Telcordia software process specialists launched a pervasive quality initiative that set the standards for all future software development. After initial standards were developed, certification became the next goal and rallying point for the organization. This article describes Telcordia's journey to International Standards Organization 9001 certification and experience and successes with the Software Engineering Institute Capability Maturity Model (CMM).

Pitts, J. F., "Integrated Process Capability: A Quantum Leap in Return on Investment", Thirteenth Annual Software Technology Conference, Salt Lake City Utah, 29 April - 3 May 2001.

An integrated process capability developed at Northrop Grumman resulted in a large increase in return on investment. This capability integrated standard platforms and network architectures, standard tools, tools to process enablers, processes, and metrics into one infrastructure that created efficiency and further improvements. This presentation describes this infrastructure and how it evolved over time at Northrop.

Porter, R., and DeToma, D., "A Mult-Site Software Process Framework, *CrossTalk*, Volume 12, #10 (October, 1999), pp. 20-25.

Today, software process leads to the new paradigm, "Better, faster, cheaper — through continuous software process improvement (SPI)." However, developing a standard software process might be considerably easier than the task of rolling it out to multiple locations across the U.S. and perhaps overseas, a concept we refer to as "zero geography." GTE Government

Systems Corp. (GSC) successfully implemented its SPI program across North America using a framework based on the 3 C's: commitment, continuity, and communications. With zero geography, GSC was able to leverage its existing assets, accelerate schedules, and minimize investments while reaping the full benefits of SPI.

Poulin, J., Measuring Software Reuse, Addison-Wesley, Reading, MA, 1997.

This book explains the most important issue in reuse measurement - defining what to count as reuse, how to count it, and why. Without a uniform understanding of what to count, all reports of reuse levels and benefits become automatically suspect. By addressing this issue, this book puts reuse measurement into a reliable and consistent context. Furthermore, it emphasizes a fundamental truth in software reuse: Business decisions drive reuse.

Pressman, Roger S., Software Engineering; A Practitioner's Approach (Sixth Edition), McGraw Hill, New York, 2005.

This has been one of the best selling guides to software engineering for both students and industry professionals for the past twenty-five years. This sixth edition is organized in five parts: (1) The Software Process, including prescriptive and agile process models, (2) Software Engineering Practice, with emphasis on UML-based modeling,(3) Applying Web Engineering, (4) Managing Software Projects, and (5)Advanced Software Engineering Topics, including formal methods, cleanroom software engineering, component-based approaches, and reengineering. There is also a web site designed to provide valuable resources for students, instructors and professionals

Reifer, Donald J., Making the Software Business Case, Addison-Wesley, Boston, MA, 2002

This book shows how to build an effective business case when one needs to justify and persuade management to accept software process improvements. Based on real-world scenarios, this book covers common situations where business cases analyses are required and explains several techniques which have proven successful in the past. The book provides examples of successful business cases and methods to present management with compelling reasons for adopting software process improvement activities.

Reifer, Donald J., "Profiles of Level 5 CMMI Organizations", *Crosstalk*, Volume 20 #1 (January 2007), pp. 24-28

Many firms that have achieved Level 5 using the Software Engineering Institute's Capability Maturity Model®Integration (CMMI®) have taken a different tact in justifying their process improvement initiative's budget. This article summarizes the profiles of high maturity organizations and explains how they go about justifying their budgets. The article also provides insight into the differing tactics that these firms use to win the battle of the budget and the reasons for them.

Richter, A., "Quality for IT Development and IT Service Operations: CMMI and ITIL in a Common Quality Approach", European Software Engineering Process Group, London, UK, June 2004.

This presentation describes the "return on quality" as DB Systems, a German company, advanced from CMM Ledvel 2 to CMMI Level 3. It was found that the cost per function decreased substantially as the company made this progression.

Rooijmans, J., Aerts, H., and van Genuchten, M., "Software Quality in Consumer Electronics Products", *IEEE Software*, January, 1996, pp. 55-64.

Despite the rapidly growing size of the software included in consumer electronics products, manufacturers must keep the number of software defects in the field at zero to avoid financial disaster. The authors describe the process-improvement efforts they undertook to achieve this increasingly challenging goal.

Rostaher, M. and Hericko, M., "Tracking Test First Pair Programming - An Experiment", *Proceedings of XP/Agile Universe 2002*, 2002.

The authors ran an experiment where a group of professional programmers working in pairs and a control group programming alone implemented a small system from pre-defined requirements. The comparison of the control group of individuals and the group programming in pairs spent almost the same time to complete the tasks. The authors believe that a more detailed research study apart from evaluating test-first programming is needed to compare solo programming with pair programming in the investigated group.

Rozum, J. A., "Concepts on Measuring the Benefits of Software Process Improvement," (CMU/SEI-93-TR-09, ESC-93-TR-186). Pittsburgh, Pennsylvania: Software Engineering Institute, Carnegie Mellon University, June 1993.

This report describes some concepts that organizations can use to develop a method for determining the benefits they have received from software process improvement (SPI) activities. Determining how SPI has financially impacted an organization is a difficult process because many factors, such as customer satisfaction from improved product quality and thus new product sales for a company, are difficult to measure.

The author describes a SPI benefit index (the ratio of dollars saved from an SPI program to the cost of the SPI program) for measuring the benefits of an SPI program. The article discusses how an organization can measure the costs (nonrecurring and recurring) associated with investing in SPI. The article then provides some methods for quantifying the dollar savings associated with increased productivity of the software staff, early error detection of software requirements errors, reduced rework costs due to an overall reduction in errors, reduction in maintenance work and the elimination of process steps as an organization's maturity increases.

Key measurements to collect to quantify SPI benefits are identified: staff hours, errors and size. Various methods of collecting this data and expanding beyond this basic set are then briefly discussed.

Schatz, B., and Abdelshafi, I., "Primavera Gets Agile: A Successful Transition to Agile Development" *IEEE Software*, Volume 22, Issue 3 (May-June 2005) pp. 36-42.

Primavera Systems provides enterprise project portfolio management solutions that help customers manage their projects, programs, and resources. When we decided to improve how we build software and increase the quality of life for everyone on the team, we found our answer in agile software development. Adopting agile practices is a process of continuous learning and improvement. Primavera's development team is a model for others looking to adopt agile processes.

Scott, W., "The Business Benefits of CMMI at NCR Self Service", European Software Engineering Process Group, London, UK, June 2004.

This presentation describes the improvements made at NCR Self Service in Dundee, Scotland as they advanced from CMM Level 1 to CMM Level 3 to CMMI Level 2 to CMMI level 3. There were noted reductions in project cycle time as the company progressed through levels. The presentation also addresses cultural issues that challenged progress and how the company overcame them.

Shelton, G., "CMMI - The Current State", CMMI Technology Conference, Denver, CO, Nov 2003.

This presentation describes the results of several Raytheon sites from using the CMMI for process improvement. The sites realized improvements in ROI, early defect containment, earned value, and rework costs.

Sherer, S. W., Kouchakdjian, A., Arnold, P. A., “Experience Using Cleanroom Software Engineering,” *IEEE Software*, May, 1996, pp. 69-76.

This article describes the authors’ experiences with Cleanroom Software Engineering at the US Army’s Life Cycle Software Engineering Center at Picatinny Arsenal, New Jersey. The authors describe the major costs of the technology transfer to the development team: classroom training and coaching. Significant increases in the team morale and communication resulted from applying this method. Productivity increases, cost breakdown, quality improvements and ROI are detailed in this paper.

Singh, R., "CMMI & Process Improvement ROI & CAR at Level 2", Fourth Annual CMMI Technology Conference and User Group, Nov 2004.

This presentation describes assistance provided by the Software Engineering Institute (SEI) to the aerospace business unit of a large company. Projects at the business unit were running over cost and schedule with poor quality. The business unit achieved Capability Maturity Model for Software (CMM-SW) Level 2, achieved Level 3, and then migrated to Capability Maturity Model Integrated (CMMI). They adopted Value Based Management and Causal Analysis and Resolution. These improvements shifted the culture, reduced defects by 44 percent, and saved $1.6 million per year in mechanical engineering.

Stapleton, J., *Dynamic Systems Development Method: The Method in Practice*, Addison-Wesley, Harlow, England, 1997.

The Dynamic Systems Development Method (DSDM) is about people, not tools. It provides a framework of controls and best practice for rapid application development. It was created by a consortium of organizations and since its publication in January 1995, it has proved to be extremely effective in delivering maintainable systems which match the needs of the business better than those produced using traditional life cycles. This book provides practical guidelines on the implementation of key elements of the method, clear recommendations for the roles and responsibilities of the members of the development team, advise on which type

of application is most likely to benefit from the method, eight lengthy case studies and numerous examples and anecdotes.

Strassman, P. A., *The Business Value of Computers*, The Information Economics Press, New Canaan, CT, 1990.

This book examines evaluating the value of Information Technology (IT) to business. The author derived a new value-added metric, Return-on-Management (ROM), and argues that ROM is a more suitable measure than Return on Investment (ROI) or Return on Assets (ROA) in evaluating investments in MIS because ROM focuses on the productivity of management, the principal user of computers.

The following quotes about risk analysis made by the author are significant to the analysis performed in my paper: "By making the risks of technology more explicit, you create a framework for diagnosing, understanding and containing the inherent difficulties associated technological and organizational innovation," "The best way to avoid failure is to anticipate it," and "Risk analysis is the correct analytical technique with which one can examine the uncertainty of Information Technology investments prior to implementation."

The author also noted that the business value of IT is the present worth of gains reflected in business plans when you add IT, which equals the difference of the business plan when you add IT and business plan without changes to IT.

Some other measurements of business value discussed by the author include gains in market share, better prices, reduced inventories, or highly motivated employees. Senior executives must compare IT investments with other uses of money.

Stutzke, R., *Estimating Software-Intensive Systems*, Pearson Education, Inc., Upper Saddle River, NJ, 2005.

Many software projects fail because their leaders don't know how to estimate, schedule, or measure them accurately. Fortunately, proven tools and techniques exist for every facet of software estimation. *Estimating Software-Intensive Systems* brings them together in a real-world guidebook that will help software managers, engineers, and customers immediately improve their estimates -- and drive continuing improvements over time.

Dr. Richard D. Stutzke presents a disciplined and repeatable process that can produce accurate and complete estimates for any project, product, or process, no matter how new or

unusual. Stutzke doesn't just describe formal techniques: He offers simple, easy-to-use templates, spreadsheets, and tools you can start using today to identify and estimate product size, performance, and quality—as well as project cost, schedule, and risk reserves

Tockey, Steve, *Return on Software – Maximizing the Return on Your Software Investment*, Addison-Wesley, Boston, MA, 2005.

This is a book about making software technical choices in a business context. According to the author, most software professionals do not know how to consider the business aspects of their software decisions, nor do they understand the importance of business aspects. However, business consequences should play a role in choosing projects, choosing a development method, choosing algorithms and data structures, and determining the amount of testing to be done. He includes chapters on inflation, depreciation, tax considerations, and other factors that should be addressed in making choices.

Tower, J., "Investment bank Technology Examples of CMMI Benefits", Fourth Annual CMMI Technical Conference and User Group, Denver, CO, November, 2004.

This presentation describes benefits obtained by software process improvement at JP Morgan, a large investment bank. Fifty three out of 59 groups are at Level 2 in the Software Engineering Institute (SEI) Capability Maturity Model Integration (CMMI), and two groups are at Level 3. Quantitative data is presented on changes in schedule slippage, defects, Return on Investment, and productivity. Implementation of CMMI Level 2 processes and deliverables satisfy four out of six Information Technology (IT) control objectives required by Sarbanes-Oxley.

Tukey, J.W., *Exploratory Data Analysis*, Addison-Wesley, Boston, MA, 1977.

The approach in this introductory book is that of informal study of the data. Methods range from plotting picture-drawing techniques to rather elaborate numerical summaries. Several of the methods are the original creations of the author, and all can be carried out either with pencil or aided by hand-held calculator.

Urioste, M., "Tomahawk Cruise Missile Control: Providing the Right Tools to the Warfighter", Crosstalk, Volume 17 No. 9, September 2004, pp. 8 - 10.

With fast-changing targets, unconventional enemies, and shadowy, pop-up targets of opportunity, our warfighters require the very best software solutions that take advantage of newest-generation cruise missile capabilities. The Tactical Tomahawk Weapon Control System gives the United States' and the United Kingdom's naval warfighters the right tools to carry out today's demanding strike missions.

van Solingen, R., "Measuring the ROI of Software process Improvement" *IEEE Software*, Volume 21, Issue 3 (May-June 2004) pp. 32-38.

Software process improvement (SPI) has been on the agenda of both academics and practitioners, with the Capability Maturity Model as its de facto method. Many companies have invested large sums of money in improving their software processes, and several research papers document SPI's effectiveness. SPI aims to create more effective and efficient software development and maintenance by structuring and optimizing processes. SPI assumes that a well-managed organization with a defined engineering process is more likely to produce products that consistently meet the purchaser's requirements within schedule and budget than a poorly managed organization with no such engineering process. We discuss about the measuring the ROI in software process improvement.

Verma, D., "The Value of Systems Engineering: Some Perspectives from Commercial Industry", Systems Engineering: A Retrospective Review and Benefits for Future Air Force Systems Acquisition, 27-28 February 2007, National Academies.

This presentation describes how systems engineering and architecture techniques can be applied to software projects, and the results of employing systems engineering and architecture techniques to 62 software projects at IBM. The projects for which systems engineering and architecture techniques were used had significantly greater productivity (and lower costs), shorter development times, and fewer defect hours during warranty than for the projects where these techniques were not used. The presentation also describes work in progress at Nokia to use systems engineering techniques for their software development projects.

Violino, R., "Measuring Value: Return on Investment - The Intangible Benefits of Technology Are Emerging as the Most Important of All," *Information Week*, Issue 637 (June 30, 1997), pp. 36-44.

Although not an article from a technical journal, this article makes some interesting points about calculation of ROI. This article grapples with establishing ROI measures for use of Information Technology (IT). Since measuring ROI is so difficult to IS managers, some new "intangible" ROI measures are starting to appear: product quality off the assembly line, customer satisfaction after an interaction, and faster time to market - these measures reflect, the author contends, a company's real sources of value and are what customers truly care about. This article discusses an approach discussed extensively in my paper - performance of a risk analysis to estimate ROI.

The author believes "There's a need for new metrics that go beyond the traditional industrial age measures that focus on cost analysis and savings." The author polled 100 IT managers to understand the importance of ROI calculations in IT investments in their organization. Of those polled, 45% require ROI calculation, 80% say ROI is useful, only 20% have formal ROI measures, and 25% have plans to adopt ROI measures in the next 12 months.

Vu, John D., "The Process Improvement Journey of Boeing Information Services, Wichita", SEPG 2005, Seattle, WA, March 8, 2005.

A presentation by Boeing about software process improvement lessons learned. Presentation explains why SPI is important, common reactions and misconceptions, and a look at the benefits Boeing experienced after implementing several improvements. Presented at the SEPG 2004.

Walter, L., "Software Process Achievement", The 1999 Software Engineering Symposium, Pittsburgh, PA, 30 August - 2 September 1999.

The Oklahoma City Air Logistics Center (ALC) has observed significant achievements in improving software processes. This presentation reviews successful process improvements undertaken and the impacts they have had on the organization as a whole. Benefits observed include increased productivity, reduced maintenance and rework, improved schedule performance and increased customer satisfaction.

Webb, D., and Humphrey, W.S., "Using the TSP on the TaskView Project", *Crosstalk*, Volume 12 #2 (February, 1999), pp. 3-10.

This article reports the first results of using the Team Software Process (TSP)™ on a software-intensive system project. The TSP was developed by the Software Engineering Institute (SEI) to guide integrated teams in producing quality systems on their planned

schedules and for their committed costs. The TaskView team at Hill Air Force Base, Utah used the TSP to deliver the product a month ahead of its originally committed date for nearly the planned costs. Because the engineers' productivity was 123 percent higher than on their prior project, they included substantially more function than originally committed. Testing was completed in one-eighth the normal time, and as of this writing, the customer has reported no acceptance test defects.

Weller, E. F., "Lessons Learned From Three Years of Inspection Data," *IEEE Software*, September 1993, pp. 38-45.

This article describes inspection experiences at Bull HN Information Systems Major Systems Division. The article describes the major metrics collected during the inspections. Defect detection rates are summarized. Lessons learned include data collection principles, metric naming conventions, ideal inspection team size and effectiveness profiles. The efficiency of defect removal is summarized and four project experiences are analyzed.

Wezka, J., Babel, P., and Ferguson, J., "CMMI: Evolutionary Path to Enterprise Process Improvement", *Crosstalk*, Volume 13 #7 (July 2000), pp. 8-11.

This is a useful introduction to the CMMI and the rational behind it. The authors acknowledge the many sponsoring agencies for this effort and show why it was needed. They also describe the CMMI product suite and provide ideas for organizations to transition from other methods to using the CMMI.

Weszka, J., "Lockheed Martin Benefits Continue Under CMMI", Fourth Annual CMMI Technology Conference and User Group, Nov 2004.

Lockheed Martin has been pursuing software process improvement to achieve Level 5 in the Software Engineering Institute (SEI) Capability Maturity Model - Integrated (CMMI), among other initiatives. This presentation describes benefits obtained at four Lockheed Martin organizations, namely Systems Integration (Owego, NY), Maritime Systems & Sensors â€" Tactical Systems (Eagan, MN), Maritime Systems & Sensors â€" Radar Systems (Syracuse, NY), and Maritime Systems & Sensors - Undersea Systems (Manassas, VA). Metrics discussed include cost and schedule performance indices, productivity, and defect density.

Williams, L., Kessler, R., Cunningham, W., and Jeffries, R., "Strengthening the Case for pair Programming," *IEEE Software*, Volume 17, Issue 4 (July – August 2000), pp. 19-25.

The software industry has practiced pair programming (two programmers working side by side at one computer on the same problem) with great success for years, but people who haven't tried it often reject the idea as a waste of resources. The authors demonstrate that using pair programming in the software development process yields better products in less time-and happier, more confident programmers.

Wohlwend, H., Rosenbaum, S., "Software Improvements in an International Company," *Fifteenth International Conference on Software Engineering*, 1993, pp. 212-220.

This article summarizes software process improvements at Schlumberger's Laboratory for Computer Science (SLCS) which began in 1989. Seventy-six organizations and 2000 developers are involved in software development. Their SEI assessment identified improvements required in project management, process definition and control, and project planning and control.

SLCS has focused on the improvements identified by the SEI, as well as requirements management, software project tracking and oversight, configuration management and quality assurance. One interesting observation made by the authors is that it is very difficult to institute change and meet existing schedules. The authors point out that the organizations with which an improving organization interacts also need to improve.

Yamamura, G., Wigle, G.B., "SEI CMM Level 5: For the Right Reasons," *Crosstalk*, Volume 10 #8 (August 1997), pp. 3-6.

The Boeing Space Transportation Systems (STS) Defense and Space Group's process improvement efforts are documented in this article. What is particularly interesting about this article is that the STS had created a CMM Level 5 organization before they adopted the CMM; and thus adopted a process based improvement program before they had to. This organization had also been collecting process related data for 15 years. Documented results include defect reduction, increased productivity, cycle time reduction, high product quality, excellent performance, high customer satisfaction, satisfied employees. Of most interest for my paper is the fact that the authors document that from employee satisfaction perspective, employee satisfaction grew from 74% to 96% because of the improvements.

Appendix A: Instructions for Use of The DACS ROI from SPI Spreadsheet Model

A.1 Introduction

Attached to this State of the Art Report (SOAR) is a diskette titled "The DACS Return-On-Investment (ROI) from Software-Process-Improvement (SPI) Spreadsheet Model." This diskette contains one Microsoft Excel®, Version 5 spreadsheet titled ROI.XLS. This appendix describes how to use the model to perform your own analysis. Instructions in **Bold** type pertain to those parts of the spreadsheet that you would change to incorporate your own estimates and experience data. We recommend that you first make a backup copy of this file.

This spreadsheet can be used for

- Software size estimation
- Software cost estimation
- Return on investment (ROI) estimation from SPI

On opening the spreadsheet from the Excel® application, you will see that this spreadsheet has 16 individual sheets:

(1) "WBS" - this sheet is used for defining a Work Breakdown Structure for a project.

(2) "Parameters" - this sheet includes various parameters and tables used for software size estimation.

(3) "Camera to Video" - a sample module size estimating spreadsheet.

(4) "Video Enhancement" - another sample size estimation spreadsheet.

(5) "Video Display" - another sample size estimation spreadsheet.

(6) "Sound Enhance" - another sample size estimation spreadsheet.

(7) "UNUSED" - another sample size estimation spreadsheet.

(8) "Summary" - consolidation and summarization of all size estimates.

(9) "COCOMO P's" - a COCOMO cost and schedule estimation worksheet.

(10) "Schedules" - project schedule worksheet.

(11) "Estimates" - Size, Cost and Schedule summary sheet.

(12) “Inspections” - ROI analysis for Fagan Inspections.

(13) “Reuse” - ROI analysis for software reuse.

(14) “Cleanroom” - ROI analysis for Cleanroom development.

(15) “SPI” - ROI analysis for a complete software process improvement program.

(16) “ROI Summaries” - A consolidated view of ROI from SPI.

Sheets (1) through (8) are used for software size estimation. Sheets (9) through (11) are used for cost and schedule estimation. Sheets (12) through (16) are used to perform return on investment analysis. Although all 16 sheets interact, it is possible to only utilize the size estimation, or the cost and schedule estimation, or the ROI estimate portions of the spreadsheets.

As a general reference for any questions about terminology and approach utilized in this spreadsheet, please see either *Pressman* [3]or *Boehm*[4].

A.2 Software Size Estimation

Before a cost estimate can be calculated by this spreadsheet, an estimate of the software size, in lines of code (LOC), needs to be established.

The first step in this process is to identify the requirements for the system and identify all tasks required (both software and non-software related) to develop the system. The list of requirements and tasks should then be hierarchically broken in to a task list, titled a Work Breakdown Structure, or WBS. A sample WBS is provided for a fictitious project in the sheet “WBS”. **You can erase all contents in this table and then develop and enter your own WBS.** This WBS has 5 columns:

(1) “WBS #” - this column is actually 4 separate spreadsheet columns in which a unique number is assigned to each task.

(2) “Task Description” - a description of the task or requirement is identified.

(3) “Source Document Code” - identifies which document contains this requirement.

(4) “Paragraph Number in Source Document” - identifies which paragraph in the source document contains the requirement.

[3] Pressman, R., “*Software Engineering: A Practitioner’s Approach*”, Fourth Edition, McGraw-Hill, 1997
[4] Boehm, B., “*Software Engineering Economics*”, Prentice-Hall, 1981

(5) "Comments" - include any comments or notes here for the task. You may wish to identify your assessment of the risks associated with this task.

The next sheet is titled "Parameters," and contains 5 tables. You should review and modify as appropriate to your organization. Since most of the factors shown on this sheet include industry averages, if you do not have history to suggest otherwise, you should utilize the factors provided in Tables 1 through 4. **Table 5 must be updated by the reader for your particular project.**

(1) "Table 1 - SLOC Language Conversion Chart" contains some industry standard lines of code conversion factors. If you are translating some software from one programming language to another, you can utilize these handy conversion factors for estimating new program sizes.

(2) "Table 2 - Memory Requirements and Productivity by Language" will have to be estimated by your system engineers. This table identifies for your processor what the average memory requirements are for one line of source code. This table is also where you can indicate your staff's productivity with each language in SLOC per day.

(3) "Table 3 - Equivalent Effort Factor" includes some additional industry standard factors for cost estimation. This shows that if developing code is 1X the effort, then reusing code requires 0.3X or 30% the effort and reusing with modifications requires 0.6X or 60% the effort. If your organization's experience is different, you should modify here.

(4) "Table 4 - Allocation of Effort" will be utilized by the COCOMO cost estimation spreadsheet and identifies the average amount of total development effort dedicated to each phase of the project.

(5) "Table 5 - Product and Component Matrix" is utilized by other sheets and identifies the breakdown of your system into major products and components within each product. **For each major <u>product</u> of your system, you should enter the name of the <u>element</u>, the WBS number that applies, the SOW number that applies, and up to 10 <u>component names</u> for each <u>element</u>. This table supports up to 5 <u>elements</u> and 10 <u>components</u> per element.**

The next 5 sheets: “Camera to Video,” “Video Enhancement,” “Video Display,” “Sound Enhance,” and “UNUSED” are size estimating sheets containing fictitious but representative data. Each sheet corresponds to each of the major elements identified in Table 5 on the “Parameters” sheet. Each of these 5 sheets is structured identically. The product name, component names, WBS number and SOW number appear automatically. Each of the 5 sheets contains 2 tables and a graph:

(1) **Within the “Actual Size Table,” you should enter SLOC estimates for each component. Enter a minimum, maximum and expected SLOC.** Minimums and maximums should be selected such that the range, with a 95% probability or a 3 sigma, covers the actual size of the component. The spreadsheet then computes a “likely” value for that component as $(\min imum + 4 * \exp ected + \max imum) / 6$. Estimated memory requirements are computed based on Table 2 of the “Parameters” sheet.

(2) **In the “Equivalent Size Table,” you divide and enter the SLOC for each component into new code, reused code or modified code.** The spreadsheet computes equivalent code sizes based on these subdivisions.

(3) A “Reuse Chart” is shown providing an assessment of the amount of reuse being accomplished on each product.

The final size estimating sheet, “Summary,” totals and summarizes graphically the size estimates for the entire system.

A.3 COCOMO Cost Estimation

The next 3 sheets: “COCOMO P’s,” “Schedules,” and “Estimates” are the sheets used for COCOMO cost and schedule estimation. This spreadsheet uses an Intermediate COCOMO model as described in *Boehm*.

A.3.1 COCOMO P’s Sheet

“COCOMO P’s,” should be the first sheet you review. **In cell B14, select the COCOMO Mode to be used. The word “Organic”, “Semi-Detached” or “Embedded” needs to be entered into this cell based on the type of system you are estimating.**

The next fields to be entered in “COCOMO P’s” is under item ii)-e) Effort Adjustment Factors. Under the column “Selected Type”, you enter for each attribute the

strings "VL", "LOW", "NOM", "HIGH", "VH", or "EH" for Very Low, Low, Nominal, High, Very High or Extra High respectively as the assessment for each attribute. COCOMO estimates the effort and schedule associated with design, coding and integration tests. It does not account for the cost of requirements analysis, software project management, configuration management (CM) and quality assurance (QA). These factors will have to be analyzed and factored in based on your organization's experience and history. However, to address these factors in this spreadsheet, a labor adjustment for requirements analysis is provided in cell B72 (a 20% adjustment is fairly standard in the industry) with the computed value in cell C72. Project management is computed (entered) in cell C91. CM is computed in cell C98 and QA is computed in cell C99 based on % labor adjustments. Similarly, schedule estimates from COCOMO do not estimate schedule lengths. The COCOMO schedule computation is adjusted in cell C130 to account for requirements analysis.

Other factors in "COCOMO P's" that you should evaluate and adjust include the phase distribution of effort, identified in cells D32:D35, and the phase distribution of schedule, in cells B121:B124.

A.3.2 Schedules Sheet

Next, the "Schedules" sheet needs to be reviewed and adjusted. The first step is to develop a "Generic Schedule" for the development effort. In "COCOMO P's", cells D136:D141 provide suggested schedule estimates by phase for your project**. In the Generic Schedule of the "Schedules" sheet, you should manually lengthen or shorten the given schedule based on the total suggested schedule length. Within each month of each phase, you should enter a typical labor usage profile (percentage) by month. The total of each row should be 100%.**

The Effort Schedule should be shortened or lengthened to match up with the Generic Schedule. The spreadsheet will then automatically distribute labor staff days into each month. **The rows below SUB-TOTAL (ABE) will need to be evaluated for applicability to your situation.** In this particular spreadsheet, a development Head Count is computed for each month. Software Project Management is computed based on head count. QA, CM and Documentation are then computed for each month.

The Labor Schedule and Cost Schedule should be shortened or lengthened to match up with the Generic Schedule. The spreadsheet extracts the category names and % effort from the "Estimates" sheet and displays them in cells A69:B73. The spreadsheet then calculates and

distributes the labor across each month for each labor category. Category names and labor rates are extracted from the "Estimates" sheet and entered into cells A81:B85 of the Cost Schedule. The spreadsheet calculates and distributes costs across each month. A grand total cost for the project is shown in the bottom cell of the Total column.

A.3.3 Estimates Sheet

The "Estimates" sheet summarizes the findings and analysis of the COCOMO Spreadsheet.

The Key Parameters and Assumptions table lists key parameters in summarizing findings of the COCOMO estimates. Size is extracted from the size estimates sheet. **You enter anticipated productivity rate in lines of code per day. Although productivity is not a parameter in COCOMO, you are encouraged to use more than one method to estimate costs to establish reasonableness of individual estimates. You should also enter the availability of people. This parameter defines the average number of days per month that a person is available, including vacations, holidays, etc.** The COCOMO Mode used in the "COCOMO P's" sheet is displayed.

The Key Values Calculated table computes and displays various cost and schedule estimates. Schedule Length and Adjusted Schedule length are extracted from "COCOMO P's." The effort required based on the productivity rate entered above is also calculated and can be compared with effort estimates from COCOMO, with the intent of making the COCOMO and Productivity estimates agree to within 5%. Total effort and costs are also displayed from the "COCOMO P's" and "Schedules" sheet.

The COCOMO Adjustment Factors you selected are displayed in the COCOMO Adjustment Factors table.

The Process Distribution by Phase table displays % effort and % actual schedule distribution by phase and labor effort days by phase.

You then enter the Labor Distribution by Category Table. For each job category, a category name and a rate per hour (with or without overhead included) is entered. You must also estimate the % of hours, on average, that each category works on the project. An average labor rate is then computed by the spreadsheet.

A.4 Return on Investment from Software Process Improvement

The sheets titled "Inspections," "Reuse," "Cleanroom," "SPI", "ROI Summary", and "Risks" are used to perform ROI analysis. Since the "ROI Summary" Sheet includes both input parameters and summary of results, this sheet is discussed first. Background for this portion of the spreadsheet can be found in the body of this SOAR. This Section assumes the reader has read this SOAR. **No inputs are required for this analysis.** If you've completed the spreadsheet parts described in Section A.2 and A.3, no new data needs to be entered.

A.4.1 ROI Summary Sheet

Four tables are required to perform the ROI analysis:

(1) The KEY PARAMETERS table is derived from other parts of the spreadsheet and from documented values in the open literature. Project Staff Size is computed by the spreadsheet by examining the head counts in the "Schedules" sheet and finding the maximum head count in any one month. Lines of Code is extracted from the "Summary" sheet (cell F20). Average Defects per KSLOC in New Code, Average Defects per KSLOC in Reused Code, Average Defects per KSLOC in Cleanroom Code, and Software Defect Removal Efficiency are as documented in the literature.

(2) The Labor Requirements to Detect Defects Table includes Staff Hours to Resolve Design Defects for an average software project, and the multiplier to be applied to that value if the defect is to be resolved During Design, During Test or During Maintenance. 1X, 10X and 100X respectively are widely recognized multipliers.

(3) The Parameters to Estimate Rework Table describes the relative efficiency of each process improvement to detect defects by life cycle phase. The % Defects Removed Before Operational parameter shows industry average efficiencies of removing defects prior to release to users.

(4) The COCOMO Parameters from Model Table brings forward the COCOMO estimating parameters from the COCOMO sheets.

As described in the body of the SOAR (Section 4.4), summary comparisons of different process improvement methods are shown in the Method Comparison for Selected Input Values Table. Charts comparing development costs and rework costs for each method are shown beginning in cell A60 and F60 respectively.

A.4.2 Inspections Sheet

Development Effort Reduction for Inspections shows (Cell B2) the documented reduction in development labor required if Fagan inspections are followed.

Costs incurred in implementing Inspections is shown in cells A49:A53, and only involves training costs.

A.4.3 Reuse Sheet

The Detailed Breakdown of Rework Table beginning in Cell A22 provides details of the rework costs for each level of reuse. The Formal Inspections New Code column computes rework costs for the new code and the Formal Inspections Reused Code column computes rework costs for the reused code. Since reused code exhibits less defects per KSLOC, increased levels of reuse reduces rework costs.

A.4.4 Cleanroom Sheet

The tables shown here are explained in the body of this SOAR (Section 3.3.3). Cell F7 computes the costs of Cleanroom as a % of total labor requirements. Cell F12 shows the productivity increase achieved by the Cleanroom method.

A.4.5 SPI Sheet

This sheet implements Capers Jones' models of process improvement as discussed in the body of this SOAR (Section 4.1).

Rows 1 through 115 of this worksheet implement Table 4.2. Many of the elements in this table are computed by looking up values in the Process Improvement Costs per Employee table, Improvement Stages in Months table, and the Improvement Parameters table.

(1) The Process Improvement Costs Per Employee table depicts the per employee costs at each SPI stage for various size organizations.

(2) The Improvement Stages in Months table shows, for various organization sizes, the number of calendar months required for each Stage of improvement.

(3) The Improvement Parameters table shows, for each stage, the % improvement for removing defects (reduced rework), the % improvement in productivity (reduced development costs), and the % improvement in schedule delivery.

Rows 120 to 146 of this sheet implement Table 4.3. The only user changeable parameter is the Programming Language selection in Cell C24. Selecting the language is through a pull-down menu selection. When the language is selected, the LOC/Function Point value displayed is the mean value for the selected language from the bottom table of the "Parameters" Sheet. The Total Lines of Code is the "Likely" value from the first table of the "Summary" sheet, and the Total Function Points is the total lines of code divided by the LOC/Function Point factor.

A.4.6 Risks

This sheet summarizes all the secondary benefits of software process improvement. None of the tables in this sheet are linked to other portions of the spreadsheet. Please refer to Section 3.5 of this report for details on each area within this spreadsheet.