

1. Introduction

The purpose of this Revised State of the Art Report (SOAR) is to provide more insight into the details necessary to demonstrate from a business, profit and loss, and senior management perspective the benefits of improved software management using software process improvement techniques. Software process improvement 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 software process improvement. This revised SOAR updates the original thinking by examining the business implications of many secondary benefits, such as improved employee morale and higher customer satisfaction. A framework is established whereby the current methods of performing software development can be compared to any proposed improvements.

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

- ◆ Reduces Development and Maintenance Costs. The cost of implementing software improvement methods are 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.
- ◆ Improves 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 software process improvement methods reduce post-release defects to near zero. Improving customer satisfaction is shown to result in repeat customer business and an improved company image.
- ◆ Reduces Cycle Time. Improvement efforts can reduce typical schedule lengths by 30% to 40%. This may translate to higher profit because it may allow organizations to beat the competition in getting product to the field, it may result in more product purchased earlier than projected, or it may 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 our customers are concerned.
- ◆ Increases Profitability. The return on investment for software improvement is very high. Many organizations have reported a 7:1 ROI. 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 doing software improvements versus not performing the improvements highly favors performing the improvements.
- ◆ Improves Professional Staff. Software process improvement improves employee morale, and increases the confidence of developers. It results in less overtime, less crisis, less employee turnover, and an improved competitive edge. The reduction in employee turnover costs and retraining costs could pay for the improvement costs alone.

The focus of this report is on process, software process improvement (SPI) and the cost benefits that can be achieved through performance of a SPI program. This paper will examine four categories of software process improvement and the return on investment (ROI) from:

- (1) performing a complete SPI program,
- (2) utilizing formal inspection techniques,
- (3) instituting a systematic software reuse program, and
- (4) using a Cleanroom development methodology.

This report also examines some of the more detailed and secondary benefits and risks associated with software engineering initiatives.

The literature has many documented success stories of cost savings resulting from software process improvements. Section 2 of this paper summarizes the findings as reported in the literature. This section also explores other writer's 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 improved competitive position which are also discussed in Section 2.

In section 3, for each of the four classes of SPI, a spreadsheet model is developed which will allow anyone to identify likely cost savings from SPI in their organization. This spreadsheet is built upon a Cocomo cost and size estimating model. An enhanced spreadsheet is also provided to provide a framework for addressing the secondary benefits of improvements.

In section 4, generalizations about the spreadsheet of section 3 are made and conclusions are drawn from the research performed. Section 5 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 mediocre (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, a software process improvement program is needed.

We also recognize that some people do not believe 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.

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. 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, return on investment (ROI), improved productivity, and improved cycle times¹ from process improvements. The literature in this area is divided into five categories:

1. Literature that discusses the business value of software engineering and engineering management
2. Literature that presents results from organizations that have invested heavily in software process improvement and shows the collective benefits of the improvements. Literature in this area address experiences, risks, and lessons learned from applying software management techniques.

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

3. 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. A third specific area, Cleanroom Software Engineering has not received as much attention in the literature but has shown some impressive results in cost savings and quality improvements over the entire product lifecycle.
4. The literature indicates there are also many secondary benefits and other factors that need to be considered in examining and measuring software improvement from a return on investment perspective.
5. Literature that discusses risks and other factors that need to be considered.

2.1 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 software process improvement.

Johnson (1997) describes 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) examines the value of IT to business. In this 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 I propose 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.

Violino (1997) discusses ROI measures for evaluating IT. He states that management 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: 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. According to Violino, the timing of investing in IT is another intangible IT ROI factor.

Brynjolfsson (1993) examines why there is such a shortfall of evidence about productivity increases from Information Technology. Whereas since the 1970s, when corporate investments began in technology, 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; 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, it’s productivity appears to be lower than someone 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) investment in change must be made, (4) the effects of the change must be measured 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.

2.2 Continuous, Sustained Software Process Improvement

The Software Engineering Institute (SEI) at Carnegie Mellon University has developed a five-level evolutionary process model of the capabilities of software development organizations called the Capability Maturity Model (CMM). According to this model, 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 (KPA) that identify those areas on which the organization must focus on to raise its software process to that level. The CMM was produced 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.

Herbsleb (1994) provides 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 Software Process Improvement (SPI). The findings by Herbsleb, as shown in Table 1, 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 1 Improvements from Software Process Improvement (Herbsleb, 1994)

Brodman (1995) reports on many non-measurable benefits from a software process improvement program. These included improved morale by the developers, increased respect for software from organizations external to software and less required overtime. Brodman notes that some organizations looked at benefits from SPI not just in financial terms, but in terms of being more competitive (cheaper and better), improved customer satisfaction (fewer post release problems in the software) and more repeat business from their customers.

Curtis (1995) concludes that software process improvement works with a measured ROI of 6:1, a two times (2X) or 3X productivity improvement and 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.

Dion (1993) reports a 7.7:1 return on investment (\$.58 Million invested versus \$4.48 Million saved in rework costs) and a 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.

As shown in Table 2, Jones (1996) documents 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, organizations perform a formal process assessment and establish a quantitative baseline of current productivity and quality levels. In Stage 1, the management methods stage, software managers are trained in planning, sizing, estimating, tracking, measurement, and risk analysis. Stage 2 concentrates on the software development processes to be followed. The next stage 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. 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.

Stage	Focus at This Stage	Minimum Cost of SPI/Employee	Maximum Cost of SPI/Employee
0	Baseline Assessments	\$100	\$200
1	Management Methods	\$1,500	\$8,000
2	Software Process & Methodologies	\$1,500	\$3,500
3	New Tools & Approaches	\$5,000	\$25,000
4	Infrastructure/Specialization	\$1,000	\$5,000
5	Reusability	\$500	\$7,500
6	Industry Leadership	\$1,500	\$4,000
	Total SPI	\$11,100	\$48,200

Table 2: Costs of Software Process Improvement (Jones, 1996)

Depending on the size of the company, Jones believes improvement can take between 26 and 76 calendar months 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.

At Tinker AFB, Lipke (1992) reports 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, a recognition from everyone that process improvement is their job, and visible progress.

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 Lab (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.

Wohlwend (1993) reports 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.

Humphrey (1993) reports 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 improved because of these improvements

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 & 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, and 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.

2.3 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 deviation 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. Users of the method report significant improvements in quality, development costs and maintenance costs. Jones (1996b) 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 observes that with 15% defects in released products, you can never have satisfied customers.

Doolan (1992) describes 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) reports 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 the 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 states 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 reports that 12% of programmer resources was for fixing post-shipment bugs. Fagan estimates 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) utilizes 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 estimates that fixing defects in released software costs as much as 80 times more than during the specification stage. He estimates that for each

hour of inspection, 30 hours are recouped, for a yearly average saving of 16.5 staff-months. Nonquantifiable 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 estimates 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) utilizes 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 observes 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 estimate 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.

Lee (1997) documents 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%.

2.4 Software Reuse

As described by Lim (1994), work products are the products or by-products of the software development process. Work products include such things as designs, specifications, 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, 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 is 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 reports 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 variabilities 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 (Basili, 1994 and McGarry 1993) has increased the average level of reuse by 300% from ~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 ~490 staff months to ~210 staff months.

Joos (1994) describes the reuse efforts on two pilot projects at Motorola. In one pilot, 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 reuser. 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.

So with the obvious benefits of software reuse, as noted by Card (1994), why do so many reuse programs fail? 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 the resistance to change. Overcoming these cultural hindrances requires training, incentives, good management and good reuse measurement.

2.5 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; 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 2.2), 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) observes 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 observes that errors from this process are much easier to fix, stating that they take 20% the time to fix as compared to traditional software.

Hausler (1994) claims that the time spent in specification and design is greater than in traditional projects, but the time spent in test 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 claims 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.

Linger (1993) shows 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 observes that the errors in Cleanroom verification are typically simple mistakes, not design mistakes. Drastically reduced maintenance costs result from Cleanroom development. He claims that experienced Cleanroom teams with subject matter experts involved can achieve a substantially reduced product development lifecycle.

At the US Army's Life Cycle Software Engineering Center at Picatinny Arsenal, Sherer (1996) reports 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 estimates an ROI of 20.8:1 from introduction of the Cleanroom methodology, where the costs of the methodology were training and coaching² 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.

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 observes error rates of 4.3 to 6 errors per KSLOC versus 7 errors per KSLOC on traditional projects.

2.6 Secondary Benefits of Improvement Efforts

This section describes literature that discusses secondary benefits of software improvement efforts. Many such benefits have been noted in the literature:

Yamamura and Wigle (1997) states 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 state that employees take pride in their accomplishments as they dramatically reduce defects.

According to Diaz and Sligo (1997), the most significant cost benefit from Motorola's improvement efforts occurs when projects finish 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.

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 and repeat business, and decreased time to market and improved product quality translates to more dollars on the bottom line.

In McGibbon (1997), I examine the benefits of formal methods and document how, like Cleanroom software engineering, formal methods exhibit less rework after products are released than in traditional software.

Curtis (1995) suggests 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.

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

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 crisis, less overtime, more business, and higher project bonuses.

Dion (1993) states that second order effects of their process improvement efforts are improved competitive position, higher employee morale, lower absenteeism, and lower attrition. They also have less late and over budget projects.

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 result from process improvements at Hughes Aircraft.

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.

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.

Olsen (1995) discusses the “Quality Chain Reaction” as being 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.

Strassman (1990) believes that proper utilization of IT will result in gains in market share, better prices, reduced inventories, and highly motivated employees.

2.7 Risks From Software Improvement

Many believe that software process improvement is a silver bullet for successful software development. However, there is a limited amount of 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 author recognizes 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) discusses 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) 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.

3. Detailed Research

As section 2 demonstrates, there is a significant amount of evidence to show that with a properly run software process improvement (SPI) program, software development organizations can drastically improve cycle time, reduce development costs, improve quality, reduce maintenance costs, improve employee morale, and improve company competitiveness.

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 2. 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®, version 5 for the Macintosh.

Table 3 identifies many of the key parameters necessary to model ROI estimates for the SPI methods in this section. 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 improve 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, (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 (1996b) 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 3: Parameters to Software Process Improvement Model

3.1 Modeling the Cost Benefit of Software Process Improvement

SPI was shown in section 2.1 to provide a positive ROI for most software development organizations. Jones (1996a) has provided the most comprehensive model for establishing ROI from SPI. As described in section 2.1, Jones has defined the 6 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 3, Table 4 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 3. For each stage, Table 4 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 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.

	Estimated Cost To Reach Stage	No. of Months To Reach Stage	Estimated Number of Defects	Productivity LOC/Day	Schedule Length	Project Development Costs (1)	Project Maintenance Costs	ROI (2)
Stage 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: ROI Benefits of Software Process Improvement

3.2 Modeling the 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 3 defines an anticipated defect rate of 7 defects/KSLOC on 39.967 KSLOC which results in 280 estimated defects. Table 5 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:

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

Table 5 computes defects detected by phase and rework hours in each phase. As stated by Curtis (1995) and depicted in Table 5, 35% of defects induced into a system occur during the design of the software and 65% during the coding phase. Table 5 also shows that, as described by Jones (1996b), 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 (1996b) has stated that formal code inspections remove 70% of all defects remaining, whereas code walk throughs 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 (1996b) 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 5. 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.

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	2.5 Staff Hours	2.5 Staff Hours
Total Design Rework	159 Staff Hours	98 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	2.5 Staff Hours	2.5 Staff Hours
Total Coding Rework	378 Staff Hours	211 Staff Hours
Test		
Defects Found in Test	51 Defects	114 Defects
Rework Hours/Defect	25.0 Staff Hours	25.0 Staff Hours
Total Test Rework	1271 Staff Hours	2861 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	250.0 Staff Hours	250.0 Staff Hours
Total Maintenance Rework	3497 Staff Hours	10491 Staff Hours
Totals		
Total Rework	5306 Staff Hours	13660 Staff Hours
Total Rework Costs	\$206,918	\$532,752
Total Savings	\$325,834	

Table 5: Effects of Inspections on Rework

3.3 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. The second positive impact is reused code has less defects/KSLOC than new code.

These double effects are shown in Table 6. 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 6 are thus possible.

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 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
Cocoma 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 Costs	\$158,560	\$117,788	\$77,015	\$36,242
Development Effort + Maintenance	\$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 6: The Effects of Reuse on Development Effort and Rework

Jones (1996a) 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 1 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.

3.4 Modeling the Effects of Cleanroom Methodology

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 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).

	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 7: Comparison to Cleanroom Development Costs

Like software inspections, the Cleanroom methodology also results in lower rework and maintenance costs. The impact of this is shown in Table 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
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 8: Rework and Maintenance Costs in Cleanroom

3.5 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.

3.5.1 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 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 (estimate)	23 Calendar Months	18 Calendar Months
Schedule Reduction		5 Calendar Months
(Penalty) / Bonus Anticipated	(\$50,000)	\$50,000

Table 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 10.

3.5.2 Cost Benefit of Reduced Staff Turnover and Better Staff Retention

This section addresses understanding the importance of highly qualified staff within a

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

Table 10: Cost Impact of Early Delivery in Commercial Organization

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 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 to leave a company is directly 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 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 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 that noted by Yamamura and Wigle (1997). Estimated turnover ratios after improvement is 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, training costs and is 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 Key-Employee (Without Improvement), Development Costs With Key-Employee (With Improvement), Schedule Length Without Key-Employee (Without Improvement), and Schedule Length With Key-Employee (With Improvement) as shown in Table 12.

	Without Improvement	With Improvement
Project DRAT	Without Best Person	With Best Person
COCOMO Personnel Attributes		
ACAP - Analyst Capability	NOMINAL	HIGH
AEXP - Analyst Experience	NOMINAL	HIGH
PCAP - Programmer Capability	LOW	LOW
VEXP - Team Exp. with VM	LOW	LOW
LEXP - Team Exp. with 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 12: The Value of Key Contributors

In table 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 Schedule Length Without Key-Employee. It highlights the impact this individual has on delivering product earlier.

3.5.3 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 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 13: Repeat Business

3.5.4 Cost Benefit of Reduced Risk on Software Projects

Finally, this section addresses the risks and potential cost impacts of improving the software organization and risks and potential cost impacts of not improving. Strassman (1990) makes the following three statements which suggests 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 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 14: Reuse of COTS Risks

The values in the columns of Table 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 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	Not Able to Achieve Rework Reductions	\$900,000	LOW	\$45,000
4	Inability to Achieve Productivity Goals	\$750,000	LOW	\$37,500
5	Not able 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 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 impact of these risks are described and summarized in the risk assessment spreadsheet in Table 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	\$128,000
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 Awards Fees	\$50,000	HIGH	\$37,500

Table 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 method for presentation of the risks are with Weighted Risk Likelihood values. For each proposed method, the Weighted Risk Likelihood values is defined as the sum of all weighted costs for each likelihood category. As an example, the Reuse of COTS Risks from Table 14, would be represented as follows in Table 17.

Likelihood	Weighted Cost
MEDIUM	\$1,875,000
MEDIUM	\$550,000
MEDIUM	\$1,075,000

Table 17: Weighted Likelihood for COTS Reuse

This data would be graphed as shown in Figure 1 below. Management is most likely concerned with the risks of high or medium likelihood. This graph will provide management an effective way to compare risks of improvements versus no improvements.

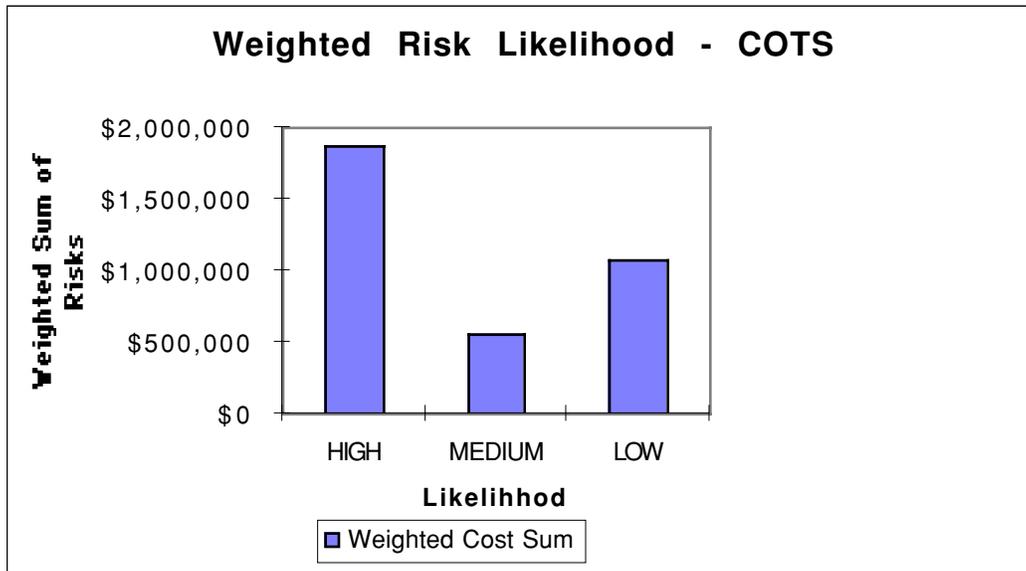


Figure 1: Weighted Risk Likelihood Graph for Use of COTS

3.6 Comparison of Results

Sections 3.1 through 3.4 address four specific software process improvement methods: SEI CMM Process Improvement, Fagan Inspections, Software Reuse and the Cleanroom Methodology. Table 18 compares the development costs, rework costs, maintenance costs, SPI costs, ROI, and savings resulting for each method. 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 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 18.

Sherer (1996) shows that the costs of the Cleanroom methodology include training and coaching costs. These costs, as shown in Table 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 SPI	\$670,949		\$22,429	\$2,344,135	\$318,358	7.48:1

Table 18: Comparison of SPI Methods

4. 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 is summarized in Table 19. The graphical representation of the Weighted Risk Likelihood is shown in Figure 2.

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 19: Comparing All the Metrics of Process Improvement

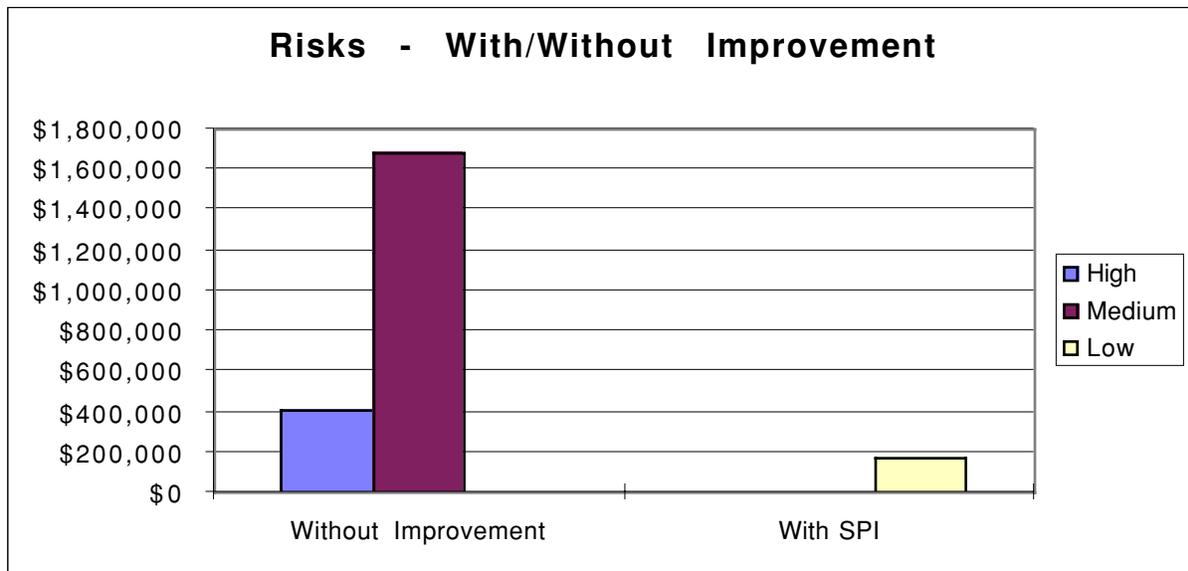


Figure 2: Comparison Weighted Risk Likelihood

4.1 The Financial Benefits of Software Process Improvement

Section 3 of this paper provides two different perspectives of the financial benefits of SPI: that of Jones (1996a) as described in Section 3.1, where a complete high level model of the cost and savings impact can be developed, and that of the specific process improvements of sections 3.2 through 3.4. It is clear from the data presented in Section 3 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 3 shows that software process improvement 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 3 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 4 compares development costs of traditional projects with other process models. Cost savings were found to be proportional to program size.

Figure shows the impact on schedule of various process models. Again schedule improvements were found to be proportional to program size.

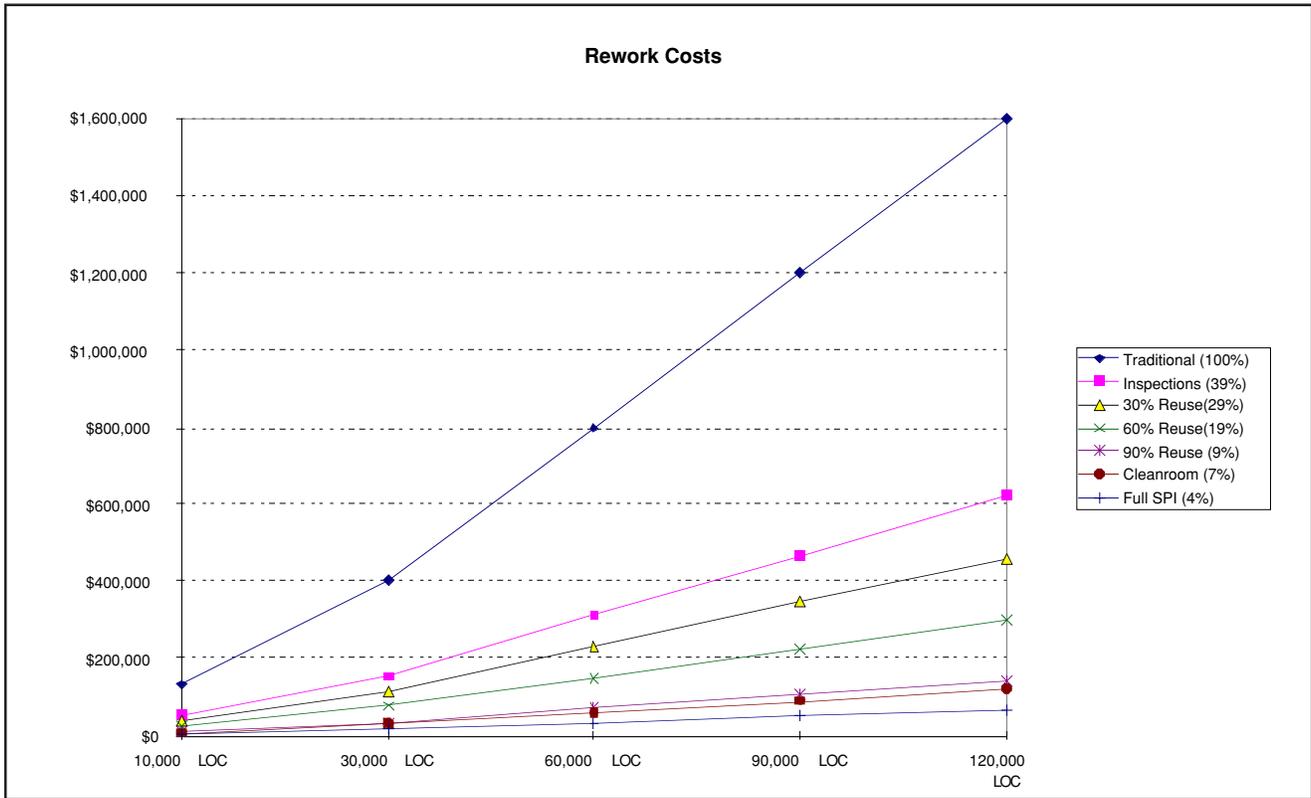


Figure 3: Rework as a Function of Program Size

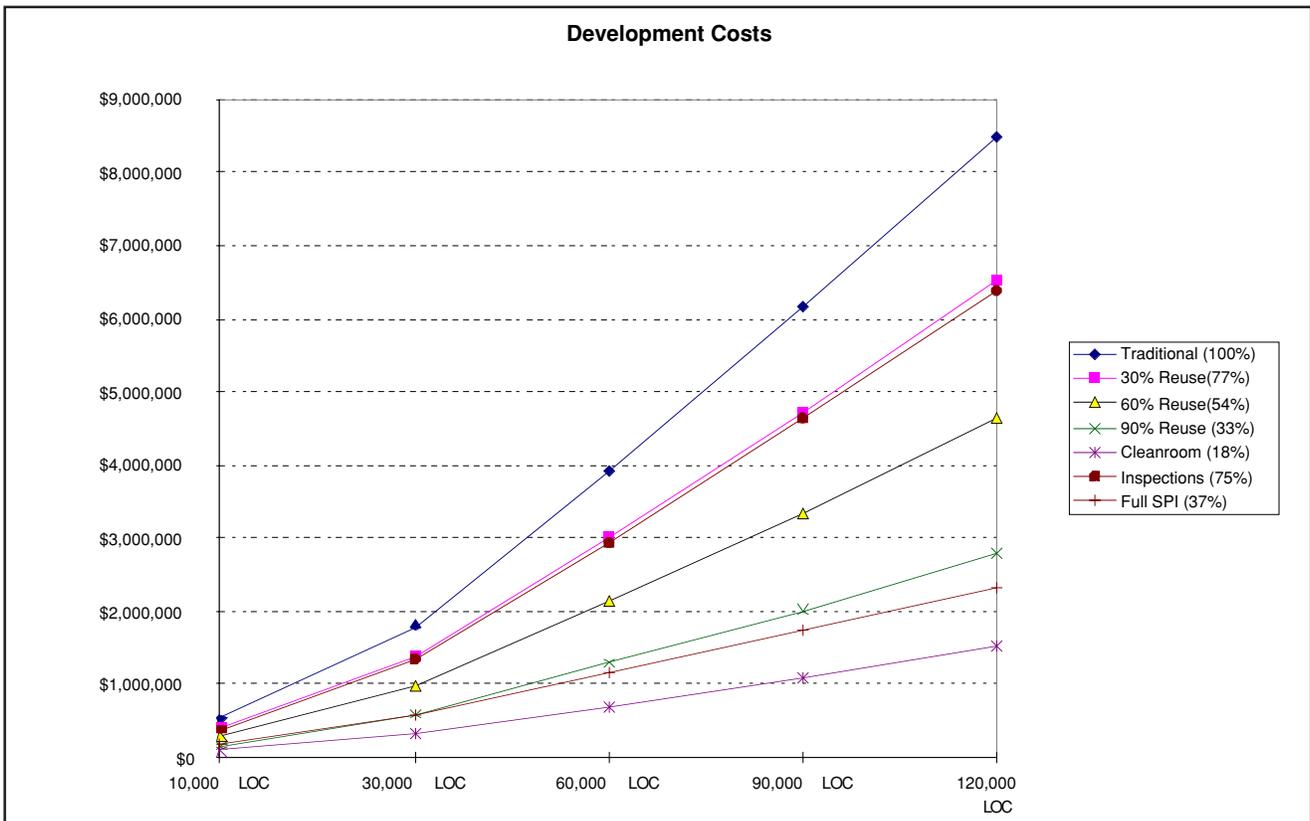


Figure 4: Development Costs as a Function of Program Size

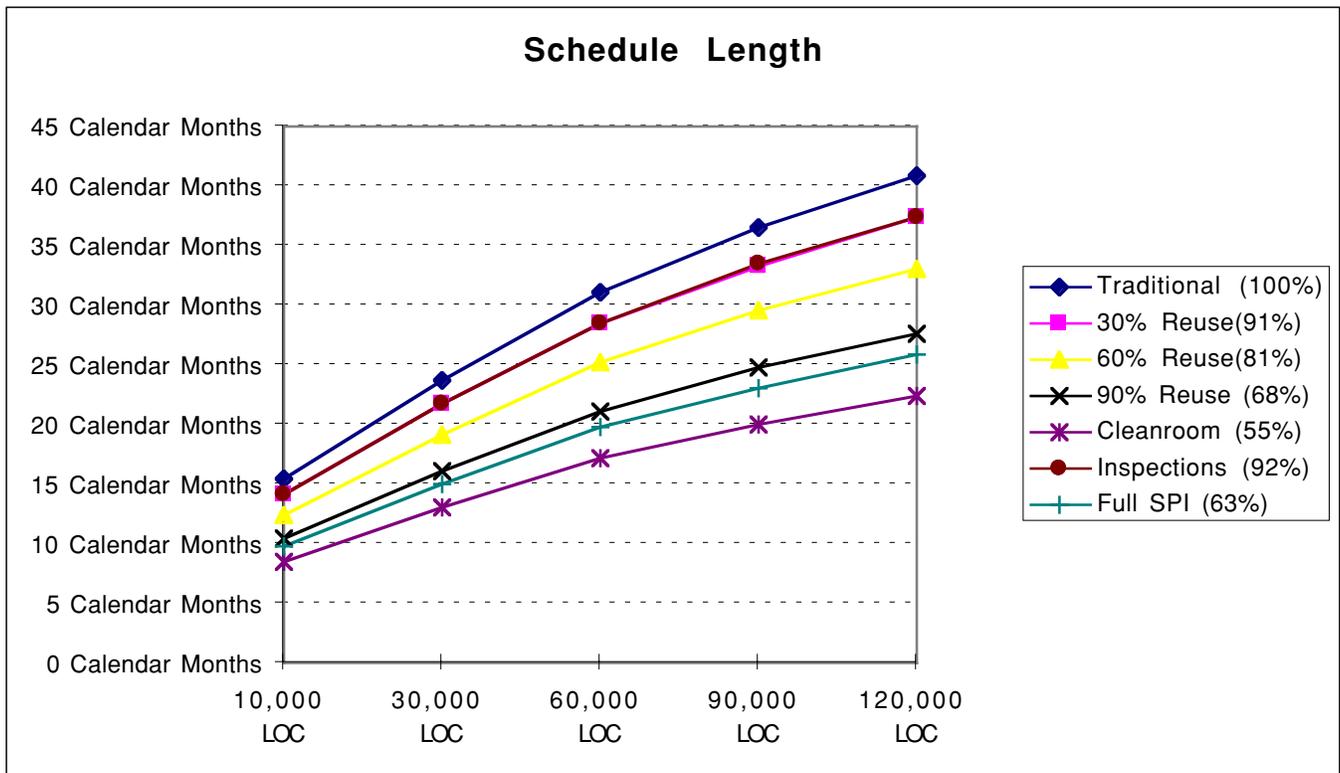


Figure 5: Schedule Length as a Function of Size

4.2 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.

5. Annotated Bibliography

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.

Boehm, B. W., “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.

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 nonmeasureable 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 mismeasurement. 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.

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.

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.

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.

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.

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, 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.

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., 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.

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.

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.

Johnson, S. T., "The Cohen Act: The Promise and the Challenge," *Crosstalk*, Volume 10 #9 (September 1997), pp. 3-9.

This article summarizes the Clinger-Cohen Act, Division E of the FY96 Defense Authorization Act, Public Law 104-106. The purpose of the Cohen Act is to realign the Government's management and investment of Information Technology (IT). The act mandates the creation of a Chief Information Officer (CIO) within every agency to ensure better technology investment, accountability and decision making. The importance of this article to my paper is in its treatment of the Government's view of return on investment. With the Cohen Act, agencies are to achieve at least a 5% decrease in the costs incurred to operate and maintain IT systems and a 5% increase in agency operational efficiency as a result of IT investments.

Examples of short- and long-term IT results by leading organizations were noted in the article: increased productivity, improved customer service, higher returns on investment, and lower risk of failure, delays, overspending. Examples of IT overruns were also noted: FAA Automation System, National Weather Services modernization program, and the IRS's Tax system modernization. The article noted that poor management practices, especially in the acquisition of software, have caused agencies to fail to reap the benefits of IT. The lack of a technical blueprint and ad-hoc and chaotic software development practices are at the root of the problem.

Jones, C., "The Pragmatics of Software Process Improvements," *Software Process Newsletter* of the *Software Engineering Technical Council Newsletter*, No. 5, Winter 1996, pp. 1-4.

In this article, the author describes the general patterns of software process improvement as taking seven stages: software process assessment and baseline, a focus on management technologies, a focus on software process and methodologies, a focus on tools, a focus on infrastructure, a focus on reuse, and a focus on industry leadership.

The author then describes the costs per employee at each stage, the cheapest improvements that can be made, the length of time to make improvements and ROI for the improvement process.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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.

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.

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.

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.

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.

Boeing's Continuous Quality Improvement (CQI) focused on productivity increase and cycle-time reduction; where some processes reduced cycle time by 50%. Initially 70% defects found during verification & 19% during validation. After peer review inspections were introduced, most defects eliminated before testing. Initially 89% defects found during development, with 11% not found; software processes now find nearly 100% of all defects. They have achieved a 97-100% award fee for more than 6 years. Initially 26% employee dissatisfaction; now employee satisfaction is 96%. 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.

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 software process improvement (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.

Please note: The PDF and HTML versions of this report does NOT include the ROI Calculating Spreadsheet. The spreadsheet is available electronically (via E-mail or floppy) for \$25 alone or FREE with the purchase of bound, hard copy of *A Business Case for Software Process Improvement Revised* technical report for \$50. These items may be purchased from the [DACS Product Orderform](#) or by calling (800) 214-7921.

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*¹ or *Boehm*².

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.
- (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. **You must update 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.

² Pressman, R., “*Software Engineering: A Practitioner’s Approach*”, Fourth Edition, McGraw-Hill, 1997

³ Boehm, B., “*Software Engineering Economics*”, Prentice-Hall, 1981

- (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 product of your system, you should enter the name of the element, the WBS number that applies, the SOW number that applies, and up to 10 component names for each element. This table supports up to 5 elements and 10 components 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 $(\text{minimum} + 4 * \text{expected} + \text{maximum}) / 6$. 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 3.5), 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 3.4). 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' model of process improvement as discussed in the body of this SOAR (Section 3.1). The SPI Model Table is as described in the SOAR. 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.

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.

Please note: The PDF and HTML versions of this report does NOT include the ROI Calculating Spreadsheet. The spreadsheet is available electronically (via E-mail or floppy) for \$25 alone or FREE with the purchase of bound, hard copy of *A Business Case for Software Process Improvement Revised* technical report for \$50. These items may be purchased from the [DACS Product Orderform](#) or by calling (800) 214-7921.