12-2000

Cost-benefit analysis for software process improvement

Daniel T. Fetzer

To the Graduate Council:

I am submitting herewith a dissertation written by Daniel T. Fetzer entitled "Cost-benefit analysis for software process improvement." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Jesse H. Poore, Major Professor

We have read this dissertation and recommend its acceptance:

Michael W. Berry, Kenneth C. Gilbert, Thomas E. Potok, Robert C. Ward

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Daniel T. Fetzer entitled "Cost-Benefit Analysis for Software Process Improvement." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Jesse H. Poore, Major Professor

We have read this dissertation
and recommend its acceptance.

Michael W. Berry

Kenneth C. Gilbert

Thomas E. Potok

Robert C. Ward

Accepted for the Council:

Interim Vice Provost and
Dean of the Graduate School

COST-BENEFIT ANALYSIS FOR SOFTWARE PROCESS IMPROVEMENT

A Dissertation

Presented for the

Doctor of Philosophy Degree

The University of Tennessee, Knoxville

Daniel T. Fetzer

December 2000

Dedication

This work is dedicated to the memory of my late brother, Dr. John W. Fetzer. My oldest brother was a role model who inspired me by his optimistic spirit and his persistence in setting and reaching high goals. He supported my goal of attaining this Ph.D. and offered frequent encouragement.

Acknowledgments

I am very grateful for the opportunity I've had to conduct this research and to work under the guidance of Dr. Jesse Poore. He has consistently provided good advice and leadership for this dissertation. Thanks also to my committee for their excellent suggestions and insight: Dr. Mike Berry, Dr. Ken Gilbert, Dr. Tom Potok, and Dr. Bob Ward.

Thanks to the many participants in the Software Quality Research Laboratory seminars for their friendship and encouragement as well as their comments and helpful discussion: Laura Prados, Dr. Stacy Prowell, Dr. Kirk Sayre, Dr. Carmen Trammel, Michael Corum, Thomas Swain, Lee Hamner, and Janet Seib.

Special thanks to Will Snipes, John Hudepohl, and Cris Kemak of Nortel Networks who provided comments, guidance, and data to support the development of the prototype tool. Nortel's financial support for this research is greatly appreciated.

I am grateful for the love and support network provided by my friends and extended family. My parents, John and Helen Fetzer, instilled the value of education and provided constant encouragement as did my brother, sister, in-laws, and others. Most of all I want to thank my loving wife, Beth, for her enduring love, understanding, and support. And special thanks to our wonderful son, Elijah, who continues to keep us entertained and helps us keep the important things in life in perspective.

Abstract

Justification of investments to improve software development processes and technology continues to be a significant challenge for software management. Managers interested in improving quality, cost, and cycle-time of their products have a large set of methods, tools, and techniques from which to choose. The implementation of one or more of these potential improvements can require considerable time and cost. Decision makers must be able to understand the benefits from each proposed improvement and decide which improvements to implement. While a variety of approaches exist for evaluating the costs and benefits of a few specific improvements such as inspections or systematic reuse, there is no general model for evaluating software process improvements.

The result of this research is a practical, useful framework to assist practitioners in evaluating potential process improvements. This general framework can accommodate a variety of methods for estimating the cost-benefit effects of a process change. To illustrate this framework a set of cost-benefit templates for Emerald and Cleanroom technologies were developed and validated. Methods for evaluating effects range from constants and simple equations to bayesian decision models and dynamic process simulations. A prototype tool was developed to assist in performing cost-benefit analysis of software process improvements.

Contents

Tables

Figures

Chapter 1

Introduction

1.1 Statement of the Research Problem

The main goal of this research is to develop a practical and useful cost-benefit analysis framework for evaluating the impact of potential software process improvements.

1.2 Motivation for the Research

Most software managers are under enormous pressure to improve the quality of their products and accuracy of plans and budgets and to reduce the cost of software projects. Such gains can be obtained by process improvements. Very few software managers are able to quantify the short- and long-term costs and benefits of contemplated software process improvements. Efforts to improve the software development process in an organization are difficult for management to evaluate in advance. Many initiatives to improve the software process require significant expenditures of resources to introduce the change and to provide on-going organizational support. Management must be convinced the proposed "improvement" supports the organization's strategic goals and will lead to positive financial impacts. The benefits from investments in software technology are often long-term,

uncertain and difficult to quantify. The costs of implementing the improvement are more immediate, more certain, easier to quantify, and can have adverse impact on short-term financial profits and on development schedules. Furthermore, management could consider several potential investments aimed at improving the software development process but with time and cost constraints that prohibit investing in the full range of possibilities. Managers need assistance in evaluating these investment alternatives.

A critical step for evaluating a potential process improvement is to estimate what effect a proposed improvement will have on the organization's bottom line over an extended period of time. Before decision makers invest into such a long-term initiative they must be able to quantify a positive impact on meeting the organization's strategic goals. A *cost-benefit analysis* (CBA) can be conducted to help decision makers evaluate such investments.

This dissertation considers issues for cost-benefit analysis of software process improvement projects and outlines a methodology for evaluating software process improvement proposals. A general framework is constructed to demonstrate an approach that can be used for evaluating any improvement. To illustrate this framework we develop cost-benefit templates for Emerald and Cleanroom process improvement technologies. A prototype tool based on this framework is designed and developed to assist the decision maker in identifying, collecting, and organizing data pertinent to the decision; exploring various "what-if" scenarios; and simulating, projecting and quantifying the net benefits to be received under each scenario.

1.3 Current Impediments

1. There is extensive literature on cost benefit analysis and on potential software process improvements. However, the literature is so extensive that it is bewildering to make all the choices and know how to proceed.

2. Existing approaches are not readily available to the decision maker.

3. Existing models require extensive data that is not generally available.

4. Data that is available may or may not be relevant to a given situation.

5. Existing models require complex simulations that require great effort to construct and parameterize, and are of dubious validity.

6. Existing approaches are more complex than necessary for most decisions, and would take more time to apply than the decision maker has available.

7. Each improvement has its own profile of effects that might vary from setting to setting; thus, a solution in one instance might not be applicable in another.

8. Benefits are often difficult to quantify and justify. Benefits are less certain, less tangible, and more long term than costs.

9. Empirical studies are often missing or inconclusive.

10. Software managers often lack the training to construct a business case with discounted cash flows and appropriate financial analysis.

1.4 Goals

1. Readily available assistance to the decision maker.

2. Support the decision for any improvement, in any environment, for any industry.

3. Involve the minimal effort and complexity that is essential to support the decision, and no more.

4. Use information that is readily available.

5. Do the best that can be done quickly.

6. Maintain an open framework that can be updated and improved as better information becomes available.

1.5 Research Contributions

1.5.1 What is Not New

1. Basic concepts of cost benefit analysis
2. Mathematics of cost benefit analysis
3. The software process improvements
4. Public and private case study data
5. Process simulation models and case studies
6. Software engineering economic models

1.5.2 What is New

1. A framework for the proposal, evaluation and decision support
2. An organized set of choices already made, each of which can be changed if compelling reasons exist
3. An organized set of data that can be used, changed or ignored
4. A dynamic simulation model of the Cleanroom software process

5. An organized set of process improvement templates that can be extended

6. An organized set of effects assigned to improvements that can be changed if compelling reasons exist

7. Specific quantification functions and models for Cleanroom and Emerald technologies

8. A tool to make cost-benefit analysis of software process improvements easy to do

9. A prospectus for a web-based service

1.6 Organization of the Dissertation

Chapter 2 reviews existing approaches to evaluating process improvements and provides an overview of cost-benefit analysis methods. Chapter 3 presents a general framework for evaluating process improvement investments. Chapter 4 gives an overview of the architecture and design of the software for analyzing process improvement investments and presents a prospectus for a web-based service. Chapter 5 describes cost-benefit templates for using *Emerald.* Chapter 6 describes the cost-benefit templates for using Cleanroom technologies. Chapter 7 summarizes the contributions of this research with directions for future activities and research.

Chapter 2

Background

This research has required study across several disciplines including software engineering, microeconomics, cost-accounting, management, and industrial engineering. This chapter provides background and definitions on specific topics most pertinent to our framework including software process, software process improvement, and cost-benefit analysis. We also review industry data related to process improvement as well as various approaches for evaluating improvements.

2.1 Software Process

The Software Engineering Institute (SEI) defines *software process* as "a set of activities, methods, practices, and transformations to develop and maintain software and the associated products, (e.g., project plans, design documents, code, test cases, and user manuals.)" [60]. Input factors to the software production process include methods, tools, training, compilers, computer equipment, and skilled labor. Large scale software development is a notoriously complex and difficult production process. Most organizations are unable to consistently produce systems on-time, within budget, and of acceptable quality.

2.2 Software Process Improvement

There are two general approaches to improving the software maturity and capabilities of an organization — a top-down framework based approach and a bottom-up goal-measurement based approach. With the framework based approach an organization seeks to emulate practices contained in quality standards or a maturity model. Commonly used frameworks are the Capability Maturity Model (CMM) [60] and the ISO 9001 standard [77]. The current practices of an organization are audited against framework requirements (e.g., the ISO 9001 clauses or key practices in the CMM) to establish a baseline. Gaps between the requirements and the current practice are used to guide improvement plans.

Goal / measurement based process improvement is a systematic approach to introducing improvements to address specific organizational problems or goals. In this approach, metrics may be used initially to answer questions and provide a baseline. Later metrics are used to verify that the introduced improvement achieved the desired goal. An example of this approach is Basili's Quality Improvement Paradigm (QIP) which uses two tools: the Goal/Question/Metric paradigm (GQM) and the Experience Factory Organization [6]. Bottom up and top down approaches are not incompatible and can be combined with good results [21]. These approaches suggest process or technology changes that *may* improve the software process of an organization. However, they do not necessarily help an organization evaluate the impact the proposed change will have on their organization.

We consider a *software process improvement proposal* to be any documented suggestion to inject technology, training, management, or process changes into a software organi-

zation for the intended purpose of improving the quality, cost, or schedule of the resulting software product. Strictly speaking, we are concerned with software *product* improvement, not just process improvement. The scope of a proposal can range from a single incremental change (e.g., introduce design reviews) to a sweeping set of changes (e.g., ISO 9001 registration). These proposals generally will have in common a goal of reducing the cost of quality or increasing productivity. Examples of potential software process improvement proposals include establishing:

- Metrics and decision support
- Independent inspections and verification
- Systematic software reuse program (e.g., domain engineering)
- Improved configuration management system

Each improvement proposal can be considered as a potential investment since the costs will be more immediate and the projected benefits will be long term. Thus the standard techniques of business investment analysis can be applied. In the case of software process improvement, the investment is the total cost involved in implementing and maintaining a process improvement. The expected higher *profit* would result from a combination of factors: a reduction in cost over the life of the software; higher sales from reduced time to market or; a higher sale price and demand from producing a superior quality product.

2.3 Current Approaches to Evaluating SPI Proposals

Just because a process improvement has been proposed does not necessarily mean it will in fact be an "improvement" to the organization when implemented. Software Process Improvements (SPIs) are not without costs and competing SPIs can provide various levels of costs and benefits to an organization. Hence, an organization needs effective ways of determining the best improvement to implement based upon their current business environment. This section reviews industry data that supports software process improvement and provides an overview of current approaches to justifying improvements within an organization.

2.3.1 Industry Data Supporting Process Improvement

A considerable amount of empirical evidence has been published on the cost and benefits of SPI efforts. Many of these studies address broad classes of improvements such as introducing Cleanroom [32] [33] [41] [54] [71], applying a GQM based improvement program [7] [54], or advancing a level in the Capability Maturity Model [23] [34]. Table 2.1 provides examples of results from these studies. There are also studies that address specific common improvements such as inspections [51] [61] [78], software reuse [46] [63], and the introduction of tools [14]. For many improvements the published studies are inconclusive, contain wide variances, or lack any hard data. For example, there are few hard numbers to support productivity or quality increases due to the use of object oriented programming, or formal methods [29] [62].

Table 2.1: Example studies on the value of Software Process Improvement

Ref.	Organization(s)	Results
[34]	Survey of 13 organizations pursuing CMM based improvement	• Annual cost of SPI per software engineer ranged from $490 to $2004. The median cost was $1375. • Productivity gain per year ranged from 9% - 67% with a median of 35%. • Yearly reduction in time to market ranged from 15% to 23% with a median of 19%. • A return on investment in SPI ranged from $4 to $8.80 with a median of $5 for each $1 invested.
[54]	NASA Software Engineering Laboratory (SEL) report on 120 projects over a 7 year period of applying Basili's Quality Improvement Paradigm.	• Cost of SPI is approximately 10% of the total software budget. • Error rate of completed software dropped by 75% (from 4.5 to 1 defect/KSLOC). • Cost of software dropped by 50 percent. • Cycle time to produce equivalent software products decreased by 40 percent.
[23] [30]	CMM based improvement at Raytheon's Equipment Division from 1988 through 1996.	• Return on investment of $7.70 for each $1 invested. • Productivity increased by 190%. • Rework costs decreased to 20% of project costs from 41%. The cost of fixing source code during integration dropped by 80% and the cost of retesting decreased by half. • Defect density decreased to 4 trouble reports per KSLOC from 17.2 per KSLOC. • Improved predictability. Reduced cost overrun from 40% to within 3%.
[32]	IBM team used Cleanroom on an AOEXPERT/MVS project which contained 107 KLOC.	• Productivity increased 36% over projected. • Error rate of 2.6 defects/KSLOC from first execution through system testing. No operational errors found in production.
[32]	Review of 17 Cleanroom projects	• Productivity improvements of 1.5 to 5.0 times over baseline projects. • Code exhibited a weighted average of 2.3 errors per KSLOC through all testing as measured from first execution vs. 25-35 errors per KSLOC for baseline development.
[71]	90 KSLOC Cleanroom project at U.S. Army's Picatinny Arsenal.	• Return on investment of 20.8 to 1. • Productivity increased 4.6 times over baseline.

There are generally three kinds of empirical results in the literature: *controlled experiments*, *case* studies and *correlational* studies.

In the software engineering literature, controlled experiments are typically done with groups of students to study various code reading or programming techniques using small segments of code. This type of experimentation can possibly reveal some insights regarding the particular techniques. However, few firm conclusions can be drawn from such experiments that can be applied to large scale development. Some notable experiments have been done in industrial settings. For example, Porter et. al. [61] conducted a long-term experiment to compare different inspection techniques on a real development project. Experiments on industrial projects are expensive to conduct and require skilled researchers to carefully define the experiments and to control threats to validity. Further, any controlled experimental results on actual projects typically analyze a small set of variables on a single project in one application domain using one language and environment. Others must replicate the experiment in other environments to gain confidence in the results—something that is rarely done due to the expense and difficulty of this type of experimentation.

Case studies typically describe the experiences of a single software organization in implementing a process improvement. Case studies are useful for showing the *potential* benefits from implementing a technology and for providing success factors and lessons learned. However, the set of case studies for a technology have a selection bias since unsuccessful attempts at implementing the technology are unlikely to be published. Also, the organizational culture, business environment, and software characteristics for a pub-

lished result may be vastly different than an organization considering the same SPI. Hence, the results do not necessarily demonstrate a general association between the improvement and the reported benefits.

Correlational studies compare data from a number of organizations and attempt to show whether a general association exists. However, even these studies are likely to have some selection bias because organizations elect to participate and provide input to the study. The majority of software organizations do not have good measurement programs in place to collect reliable, objective data to report to such studies. This is especially true for low CMM maturity organizations. Thus, low maturity organizations and organizations that lack measurement programs would be less likely to participate in correlational studies.

There are also problems in how results are reported. For example, the authors are typically proponents of the technology and tend to emphasize benefits but say little about the costs. The computation of the Return on Investment (ROI) from using the technology is often flawed. For example, there is a general failure to apply a discount rate to the cost-benefit flows. This tends to skew the results in favor of the benefits since they are usually realized later than the costs. Also, the benefit effects are often computed from a before-after perspective rather than from a with-without perspective. For example, recent results may be compared with productivity results from years earlier before the organization embarked on the improvement program. The studies typically fail to account for other factors during the time period that may have contributed to the benefit effects, such as more powerful workstations, improved development tools, or more talented developers. Finally,

the studies rarely provide quantitative data on benefits that occur outside the scope of a single project or after the release of a product. Benefits that are difficult to quantify are typically not quantified.

In spite of these difficulties, published results can be useful to support a decision for a process improvement. The ROI, productivity, and defect reduction numbers provide support for the decision as well as the success factors and lessons learned, but they do not necessarily provide an accurate or complete picture of costs and benefits that can be expected for the SPI in a specific organization.

Finally, management is self defeating. Even when a software development unit pays the price for training and technology and delivers phenomenal results, such results are ignored as the unit is dismantled to staff new projects, effect reorganization, or adjust to mergers and acquisitions.

2.3.2 Project Cost-Estimation Models

Some cost-estimation models can be used to estimate the effect of an improvement by preparing two estimates: one that represents the cost and schedule of a project without the proposed improvement, and a second estimate with the improvement. The two results are then compared to estimate the impact of the improvement. For example, the intermediate and detailed COCOMO models [9] contain a set of fifteen effort adjustment factors or cost drivers. The form of the intermediate COCOMO equation for estimating effort is:

$$PM = A(KSLOC)^B \cdot \prod_{i=1}^{15} (F_i)$$

where, *PM* is the person-months of effort, *A* and *B* are equation parameters, *KSLOC* is the estimated delivered Kilo-Source Lines of Code, and the F_i are the fifteen effort adjustment factors. Two of these effort adjustment factors relate to process and technology improvements:

- *MODP* - the use of modern program practices (MPPs) (i.e., process improvements such as inspections and incremental development), and
- *TOOL* - the use of development tools.

The nominal value assigned to each factor is one with higher values assigned for immature organizations and lower values assigned for the most mature. The suggested value for the MODP factor ranges from 1.24 for an organization who doesn't use any MPPs to 0.82 for an organization that routinely uses all suggested MPPs. This range allows a 51% increase in productivity (or decrease in project effort) based upon extensive use of modern programming practices. In the detailed COCOMO model the MODL effort multipliers can be adjusted by life-cycle phase as shown in Figure 2.1. Note that most of the cost savings for MPPs come in the later Integration and Test life cycle phases.

The more recent COCOMO II model [10] includes a scaling driver for the organization's estimated CMM based process maturity (PMAT) that would allow similar "what-if" scenarios. Using a general cost model such as COCOMO to estimate the impact of a tech-

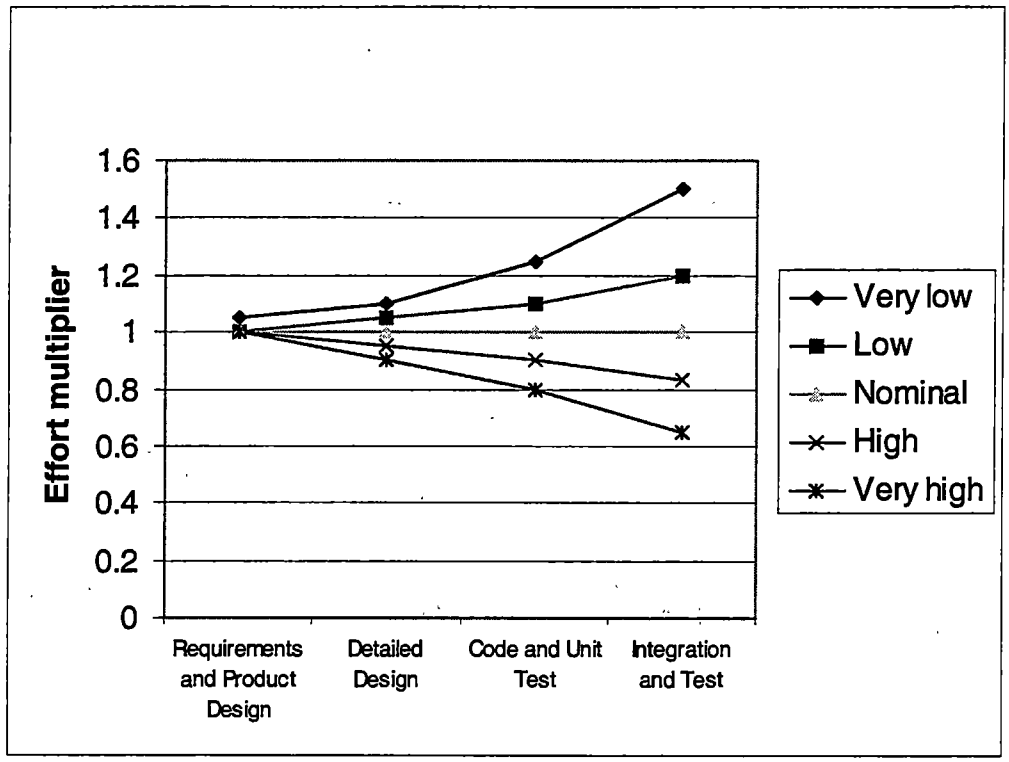


Figure 2.1: Effort multipliers by phase: Modern programming practices[a]

a. Source: [9], p. 453, Figure 27-1.

nology can be useful for determining an approximate range of cost savings. However, the decision maker has little help in determining what adjustment factors to use for specific improvements. Also, these models do not address organizational cost-benefit impacts for implementing the technology and are unable to capture other effects outside the scope of the model such as reduced maintenance, increased sales, or faster time to market.

2.3.3 Economic Models

Economic models have been developed to estimate the value of specific improvements. In particular, the literature contains many cost-benefit models for systematic software reuse

[18] [47] [52] [63]. These models typically consider the additional cost to develop reusable software components and compute the savings from reuse, the return on investment, and the number of times a module must be reused to breakeven. According to Lim [47], most of these models do not take into account the time value of money, most of them do not consider savings from the maintenance phase, most of them do not account for the overhead cost of reuse, and most do not take into account increased profit from shortened time to market.

Several of these reuse models consider how the business case for software reuse is difficult to justify under a single project view. The benefits from systematic reuse accrue over time and over a number of projects. A persistent difficulty in introducing systematic reuse is the scope of decision making regarding reuse. Project managers make decisions that optimize their current project, not future projects that could benefit from reuse. Although reuse may yield significant cost-savings for a series of projects, individual project managers have little or no incentive to incur costs and delays to make modules reusable by other projects. The reuse models of Malan [52] and others [63] help to make the value of reuse across a succession of projects more quantifiable and visible to assist higher level decision making.

Taking a multi-project, organizational view also applies to justifying other process improvements. Much of the cost of implementing an SPI involves changing the organization's culture and way of doing business. For some improvements it may take time and two or three projects in order to fine tune the process changes and to fully recover the original investment.

McGibbon [56] has prepared a useful set of economic models for estimating the effects from several improvements including CMM based software process improvement, inspections, reuse, and Cleanroom. He also provides rare examples of quantifying the less tangible secondary benefits of process improvement such as the value of improved schedules, reduced employee turnover, and improved customer satisfaction. His models, parameterized based on reports in the literature, are specific to each of the considered process improvements. His analyses do not take into account the time value of money and do not provide a direct way to be customized for a particular organization.

Economic models are useful for evaluating a specific improvement but often do not provide estimates on the full scope of costs and benefits to be considered.

2.3.4 Cost of Software Quality

Many software process improvement projects have an implicit or explicit goal of reducing the cost of quality. The cost of quality concept was originally described by Juran and Gryna [36] as those costs that would be eliminated if all workers were perfect in their jobs. The American Society for Quality Control (ASQC) has defined the following categories of quality costs [15] [58]:

- Prevention costs - incurred to prevent poor quality from being produced. For example, analysis and planning for quality, training, development of process controls. In the software world, these costs would include training practitioners in a new methodology, planning and establishing a metrics program.

- Appraisal costs - activities undertaken to prevent poor quality from being processed beyond the point at which they become nonconforming or from being delivered to customers (e.g., inspection and testing of software or design documentation).

- Failure costs - costs required to evaluate and correct or replace software not performing to specifications or failing to meet customer needs. For software, this would include the time spent analyzing and modifying source code, re-building and regression testing of executables in order to correct underlying faults.

  + Internal failure costs - associated with products that fail to meet specifications and are identified before the product or service is delivered to the customer. For example, if an integration test reveals a problem, then the cost to analyze, correct, and retest the problem code would be considered an internal failure cost.

  + External failure costs - incurred because poor quality products are delivered to customers. This category includes the cost of handling customer complaints, returns and allowances, customer ill will, product liability, and loss of future business, as well as the cost of analyzing, correcting, testing, building, installing, and documenting code patches to correct problems.

To achieve a goal of reducing the cost of quality may require increasing spending for prevention and appraisal cost in order to reduce the more expensive internal and external failure costs.

The cost of quality concept provides a convenient classification system for costs related to quality. The concept has been applied to software development by various practitioners and researchers. For example, Slaughter defines a Cost of Software Quality (COSQ) metric and a Return on Software Quality (ROSQ) metric and uses them to compute the value of four improvements at BDM International [72].

2.3.5 Software Process Simulation Models

A *software process simulation model* is an abstract representation of an actual software process that can be simulated computationally. As discussed previously, the software engineering literature provides little data that scientifically proves the effect of potential process improvements. It is extremely costly to perform controlled experiments of actual software projects and thus they are rarely done. Simulation offers an economical approach to conducting experiments on an abstract representation of a real project. These models are useful for gaining insight and understanding into the many interrelated, dynamic factors involved in producing software. However, a model is an abstraction and leaves out many details. The cause-effect relationships codified into a model are often tenuous and poorly justified. Thus the usefulness of the model depends on how well it captures the most important aspects of a real software organization.

In recent years a number of approaches have been explored for modeling the software development process including specialized languages [67], precedence networks [24], Petri nets [43], discrete simulation [31], state-based simulation [66], and system dynamics [1] [51]. Many of the existing modeling approaches are concerned with understanding or supporting the software process in an organization and do not have SPI evaluation as their primary goal [53]. This section will further examine simulation modeling work that has been used specifically for evaluating the impact of software process improvements — the state-based Task Element Decomposition approach of Raffo [66] along with various efforts using system dynamics modeling.

Task Element Decomposition (TED)

Raffo [66] adapted methods from the Operations Management literature to synthesize the Task Element Decomposition (TED) method for quantitative modeling of software development. TED uses a Markov Chain framework with the states representing different phases of the development process. Each task can be decomposed into kernel activities with random processing times. The total processing time of an operation is the sum of the times for all activities associated with the operation. TED is used in conjunction with Statemate, a commercially available process modeling tool. Raffo illustrated the TED method using a small example problem to compare the impact of inspections on the process. From his analysis he was able to compute and compare the total duration, total effort, and remaining errors for both the baseline process and the "baseline with inspections" process.

His approach is significant in that he offers a quantitative way to predict quality, cost, and schedule from within one model. However, Potok [62] argues that the software development process does not meet the conditions of a Markov Chain. Also, the model is complicated, difficult to construct, and only models a small portion of the development process.

System Dynamics

System dynamics (SD) is an approach to simulation modeling developed in the late 1950's by Forrester [27] to study the behavior of industrial and business systems. Since then, SD models have been developed to study a wide range of problems from managing research and development projects to understanding urban decay to analyzing world impacts of population growth. More recently SD has been applied to studying software development management issues.

System dynamics is based on techniques and principles adapted from control systems theory. A primary goal of SD is usually to understand the behavior of information feedback loops related to a problem. A feedback loop is a closed sequence of causes and effects, often with some delay introduced. A system dynamics model consists of a set of differential equations to model a process. The model can be simulated over time to test various alternative policies.

A system dynamics model of the Cleanroom software development process was developed as part of this research to analyze the cost-schedule impacts of incremental develop-

ment. More details on system dynamics and an overview of the Cleanroom model are in Appendix D.

Abdel-Hamid and Madnick's Model. The use of SD for understanding software development was pioneered by Abdel-Hamid and Madnick [1]. They developed a detailed, integrated model of the software development process for a project based on an extensive review of the literature combined with interviews with several software project managers. The scope of their model included personnel resources, software production, planning, and control sections. They validated their model against an actual NASA ground support software system for a satellite. They performed a variety of experiments on their model to study the interactions of various phenomena such as Brook's Law, Parkinson's Law, and the Deadline effect. They also examined the economics of software quality assurance (QA) to attempt to determine the optimal expenditure on QA. One interesting result from their experiments was that higher code writing productivity leads to an increase in the optimal percentage of effort to spend on QA activities. Unfortunately, their definition of QA lumped together requirements review, code review and integration testing. This high amount of aggregation makes it difficult to isolate the impact of specific improvements such as code inspections. Although their model is useful for understanding many software engineering phenomena, it is very complex and unsuited for quantitative assessment of cost and schedule impacts of specific process improvements in a particular environment.

Madachy's model. Madachy [51] developed a system dynamics model of an inspection-based software process and used it to investigate the impact of inspection practices on cost, schedule, and risk. His model was calibrated with data from two similar projects at Litton Data Systems except that one used inspections and the other did not. He was able to accurately reproduce the effect shown by the Litton data as well as experiment with modifying a number of parameters of the model. For example, the Litton data showed an ROI for inspections of 2.32 to 1 compared to an ROI of 2.02 to 1 for inspections obtained from model simulations. His model can also produce effort and schedule predictions based on different inspection policies.

The Madachy model is useful for estimating the effect of different inspection policies. However, it is not easy to tailor the model for a particular organization or for a particular process improvement. Many software projects do not collect the data required to parameterize the model for their organization. Madachy noted the difficulty in finding data for validating the model: "No project data was found to be complete enough for a global comparison of total effort, schedule and inspection parameters."

Problems With Existing Simulation Models for SPI Decision Support

Simulation models are often difficult to parameterize, understand, and use. The model may not match the process used by an organization and require time-consuming modifications by experienced modelers. No general model of the software development process is capable of evaluating the specific impact of any arbitrary process change that might be considered. If such a model did exist it would be too complicated for practical use. The

decision maker is unlikely to place a high confidence in the results obtained from a highly complex model that is poorly understood. Software process models tend to be single project oriented where improvements may impact the capability of an organization over many projects.

2.3.6 Other Benefit Evaluation Approaches

There are a variety of other approaches that have been reported in the literature for evaluating benefits of proposals including risk reduction and bayesian decision analysis.

Risk reduction involves identifying potential risks, the estimated loss that would occur if the risk occurred, and the likelihood that the risk will occur for the baseline ('as is') scenario. A potential SPI can be evaluated for its impact on reducing the likelihood of the risk. An example of estimating an SPI's value in reducing risk is given in Section 5.3.5.

Bayesian decision analysis is a structured approach to evaluating choices with an uncertain pay-off for those choices. Decision analysis is useful for evaluating the value of increased information on choices involved in routine decision making. An example of bayesian decision analysis is provided in Section 5.2.4.

2.3.7 Summary of Evaluation Metrics and Methods

We have reviewed a variety of approaches for estimating the benefits of software process improvement. While all of these approaches can provide useful information, none of them by themselves provide a complete picture needed to evaluate all the cost-benefit impacts

for any improvement for any organization. Further it is difficult to make useful comparisons between competing improvements or to understand the relationship among complementary improvements.

From our review of literature, various metrics are used for evaluating process improvements as summarized in Table 2.2. Although these measures are useful data points for a decision maker, no one of them is adequate as an overall measure of the value of the SPI. Estimates of productivity, quality, effort, and schedule impacts can be useful to a decision maker, but fail to provide a single criterion for evaluating the improvement. The Return on Investment is the only metric that comes close to this goal.

Table 2.2: Commonly cited SPI evaluation measures

Type	Measurement
Productivity	Increase in productivity (output per unit input). For software productivity output is typically measured in lines of code or function points while input is measured in terms of effort hours or cost.
Quality	Reduction in internal failures
	Reduction in field defects
	Reduction in error rate
Cost / Effort	Effort or cost savings realized from improvement
	Effort or cost expended to implement improvement
	Reduction in non-conformance cost
Schedule	Reduce schedule overrun and increase schedule predictability.
	Savings in overall schedule
Overall	Return on Investment. Typically cited as the cost (or effort) savings divided by the cost (or effort) to implement an improvement.
	Reduction in Cost of Software Quality

2.4 Cost-benefit analysis

This section reviews cost-benefit analysis literature from microeconomics and management literature outside the usual realm of software engineering. A *cost-benefit analysis* (CBA) is an evaluation of net benefits associated with one or more proposed alternatives for achieving a defined goal. Cost-benefit analysis is the term used by economists for the evaluation of public projects [69]. A closely related term is *capital investment analysis,* a collection of techniques for comparing and deciding between capital investment alternatives [16]. Capital investment analysis uses many of the same methods as cost-benefit analysis. The primary difference is in the scale of the problems being addressed. Because CBA was developed to evaluate large public projects it includes theory and methods for evaluating effects that may not have readily available market prices. On the other hand, capital investment analysis is typically focused on evaluating smaller, private capital investment alternatives. This research adapts the cost-benefit analysis approach of Sassone [69] with applicable investment analysis methods [16] to synthesize a method and framework to support SPI evaluation.

There are two primary ways a CBA can be used [42]:

1. As a planning tool for assistance in choosing among alternatives and allocating scarce resources among competing demands.

2. As an auditing tool for performing post hoc evaluations or follow-up studies of a previously implemented proposal.

This research focuses on the first use, but the CBA methods and framework developed here can be used for the latter purpose. Follow-up studies help show the value of past improvements and provide valuable data for future efforts.

2.4.1 Decision Criteria

A number of different methods have been suggested for comparing alternative proposals, but the Net Present Value method is considered to be the superior to all the others. This section examines three common decision criteria: Net Present Value (NPV), Internal Rate of Return (IRR), and Return on Investment (ROI).

Net Present Value (NPV)

Net present value (NPV) is a method for discounting projected costs or benefits which will occur in the future. Essentially, the NPV recognizes that money has a time value (even in the absence of inflation). For example, if a proposal is expected to yield a benefit of $100 next year, we might value that $100 next year as $90 today. The formula for NPV is

$$NPV = \sum_{t=0}^{n} \frac{B_t - C_t}{(1+r)^t} \qquad \text{(EQ 1)}$$

where

B_t is the dollar value of benefits received at time t,

C_t the costs incurred at time t,

r the discount rate,

n the life of the project, and

t is time in units such as years or months.

A proposal subjected to a CBA will typically have its costs and benefits spread over a number of years. In order to reduce the stream of costs and benefits to a single number, the Net Present Value (NPV) is computed. The NPV is examined in more detail in the next chapter.

Internal Rate of Return (IRR)

The Internal Rate of Return (IRR) is defined as the rate r used to discount the future which would make the NPV of the project equal to zero. A proposal with an IRR that exceeds a predetermined social discount rate (e.g., cost of capital) is deemed acceptable. There are three problems with this criterion:

1. The r that solves (EQ 1) is not necessarily unique. Since the equation is of degree n, it has n roots. Suppose d is the predetermined discount rate, both r_1, r_2 solve (EQ 1), and $r_1 < d < r_2$, then the IRR provides contradictory results.

2. The criterion assumes a single discount rate over the life of the project. It may be appropriate to set one social discount rate for the first few years (say d_1) and a higher rate for later years (say d_2) to account for higher risk in those years. Sup-

pose we compute an IRR (say r) between those values ($d_1 < r < d_2$). Once again the IRR provides contradictory results.

3. The NPV and IRR can give contradictory results when comparing two different proposals [16] [69] with the NPV indicating the best alternative.

Return on Investment (ROI)

The Return on Investment (ROI) (also called the Benefit-Cost Ratio (B/C) or a profitability index) is the ratio of discounted benefits to discounted costs. The formula for computing the ROI (or B/C) is

$$ROI = B/C = \frac{\sum_{t=0}^{n} \frac{B_t}{(1+r)^t}}{\sum_{t=0}^{n} \frac{C_t}{(1+r)^t}}$$

The ROI gives the discounted benefit per dollar of discounted cost. ROI is a frequently cited metric in the software engineering literature, but it has a fatal flaw when it is used to compare two or more proposals in that it doesn't take into account the size of the investment. For example, the smallest of two proposals may have a larger ROI but have the smallest total net benefit or NPV.

Another problem with the ROI calculation is that it is sensitive to whether a financial effect of a proposal is classified as an increase (decrease) in benefits or as a decrease (increase) in costs.

However, the ROI does play a role for a certain type of decision. That is when the decision involves choosing the optimal mix of several proposals subject to a capital constraint. In this case, selecting proposal with the highest ROI > 1 until the budget is exhausted will maximize the total NPV.

Although the ROI metric is frequently used in software engineering literature, the metric is seldom discounted to account for the time value based on when the benefits are received. Also, it is often inappropriate to use ROI as a comparison criterion between mutually exclusive SPIs. McGibbon [56] provides an example of where the NPV and ROI lead to different ordering in comparing SPIs as shown in Table 2.3. Note that the NPV is the superior criterion since it would provide the highest value to the company.

However, a direct comparison of Cleanroom to Inspections is also inappropriate for another reason. Cleanroom represents a broader set of methods than Inspections and only one of the Cleanroom methods — Functional Verification — directly compares with Inspections. All the other methods of Cleanroom are compatible with, and are not

Table 2.3: ROI versus Net Present Value[a]

SPI	Costs	Benefits	Net Value	ROI
Formal Inspections	$13,212	$946,382	$933,170	71.63 to 1
Cleanroom	$77,361	$2,528,372	$2,451,011	31.68 to 1

a. Excerpt of data from [56], Table 18, p. 26. The author implicitly assumed that $r = 0$.

intended to replace the role of Inspections. In McGibbon's example, Cleanroom requires higher cost to implement than Formal Inspections because more new methods are implemented. This example points to the need to better understand the relationships among competing process improvements.

2.4.2 Structuring the Decision Problem

There are three mutually exclusive forms a CBA decision problem may take:

1. Evaluate whether or not to implement a single proposal.
2. Choose a single proposal to implement from among several alternatives.
3. Select a set of proposals to implement from a larger set of possibilities.

For a simple decision problem that only involves whether or not to accept one proposal, then the decision criterion would be to select the proposal if its projected NPV is greater than zero. If choosing a single proposal among several alternatives, then select the proposal with the maximum NPV. If choosing multiple proposal from a set of possibilities then the problem is a little more complicated. In this case, one must first determine whether or not the proposals are independent and if the proposals are subject to a capital constraint which limits the initial expenditures that can be spent on the selected set of proposals. A proposal is independent of other proposals if the NPV of a proposal is not affected by whether or not the other proposals are implemented. If proposals are dependent, then one must form all possible subsets of combinations of proposals and evaluate

the NPV of each combination. Use the following algorithm to determine the decision criterion based upon the form of the decision problem.

```
SELECT form of decision problem
CASE Accept or Reject One Proposal
        Accept proposal if NPV > 0
CASE Choose One of Several Proposals
        Select proposal with maximum NPV
CASE Select a Set of Proposals
        IF proposals are independent
        THEN
                IF Capital Constraint
                THEN rank by ROI > 1
                ELSE rank by NPV > 0
                END IF
        ELSE (if proposals are dependent)
                IF capital constraint
                THEN find feasible sets maximize NPV
                ELSE find possible sets maximize NPV
                END IF
        END IF
```

2.4.3 Identifying Costs and Benefits

Identifying costs and benefits is the most important and one of the most difficult steps in conducting a benefit-cost analysis.

A cost is measured by the resources required to procure or implement some aspect of a proposal. Examples of SPI related costs include extra time to perform a new process step, consulting fees, training materials and the cost of tools to support the SPI. In general costs are relatively immediate, certain and tangible.

Benefits often take the form of cost avoidance such as reduced rework, error reduction, improved quality, time savings, reduced time to market, and improved process control. There are also less quantifiable benefits that are cited in the literature for SPI such as

improved customer satisfaction leading to higher future sales and customer retention. For SPI proposals, benefits are often more long-term, uncertain and less tangible than costs.

Benefits should be defined in specific, quantifiable terms. A vague definition of a benefit such as "improved quality" is of little value. This kind of benefit can be broken down into more specific quantifiable components, such as, reduced rework and reduced field failures.

Costs and benefits for an SPI can be identified from the literature and from considering the impacts of the SPI within a particular environment.

A preliminary matrix of costs and benefits should be created. Columns can be added for each type of stakeholder. The cost-benefit list should be reviewed to insure it is valid and to check for double counting. It must be determined to what extent each cost or benefit can be quantified. The review should determine data availability and identify what data is needed.

2.4.4 Quantifying Costs and Benefits

The second most critical aspect of conducting a CBA is quantifying the costs and benefits and determining the time periods the costs and benefits will be realized. The main difficulty in quantification is the unavoidable fact that the analyst is faced with forecasting the future. However, as much as possible, it is still important to quantify these impacts. As Sassone has stated: "Only through quantification is the aggregation of effects and the analysis of trade-offs generally possible" [69]. There is no one procedure that can be used for

quantifying effects of process improvements. However, the use of economic principles and models can help guide us to reasonable quantification approaches.

The estimator should state the source of all assumptions and estimates. The estimator should only be concerned with *marginal* cost-benefits flows (i.e., cash flow differences from the baseline scenario). The organization's historical data as well as data and estimates from the literature can be useful for estimating cost-benefit effects.

2.4.5 Setting the Discount Rate

The discount rate is a critical parameter in the NPV calculation. The discount rate can affect whether a single proposal has a NPV > 0 or change the ratings among proposals. High rates penalize proposals with benefits occurring farther in the future. A lower rate discounts the future less than a higher rate. Within a private business, the discount rate should already be established by top management based on the cost of capital for the business or the opportunity cost.

If there is concern or uncertainty about which rate to use, it may be useful to compute a *critical rate*. The critical discount rate is the rate at which the NPV calculation changes sign. If the rate is high or low, then knowing the exact rate may not be important. For example, suppose the critical rate is computed to be 18% and any rate at or below 18% results in an NPV > 0. Since you are confident the true discount rate is below 18%, you conclude that the proposal is worth implementing.

Another consideration in setting the discount rate is the risk of whether or not future benefits will actually be realized. A risk premium can be added to the rate to account for

benefits that are highly uncertain. For example, taking time now to invest in making modules reusable may not pay off if future software projects fail to reuse them. Hence, an extra factor can be added to the discount rate to account for that risk.

2.4.6 Performing Sensitivity Analysis

Some of the costs and many of the benefits in a CBA will be estimates. Such estimates may be based on a probability distribution (perhaps using a combination of subjective and objective probabilities), but the analyst must arrive at a single number to put into a CBA. The problem with using an expected value is that it does not account for society's attitude towards risk.

For example, we may believe that we have a 50% chance of receiving $0 and a 50% chance of receiving $1000 for some postulated benefit. The expected monetary value (EMV) of this probability distribution is $500. However, would you as an individual be willing to pay $500 for a lottery ticket that has 50-50 chance of winning $0 or $1000? Most people would not be willing to risk $500 for such odds.

Since society in general is adverse to risk, the appropriate value to assign is the *certainty monetary equivalent* (CME). For example, suppose the CME or the average price members of society would be willing to pay for the above lottery ticket is $380. Then society is adverse to risk and is extracting a $500 - $380 = $120 penalty for the risk present in the probability distribution. In other words, in estimating a benefit we must deduct the cost of bearing risk from the expected value of the distribution. Higher variance in a probabil-

ity distribution exacts higher costs for bearing risk and thus lowers the value of the expected benefit.

Finding the CME requires knowing society's utility function. But there is no specific procedure for determining such utility functions. In practice, sensitivity analysis is performed to estimate the degree of error in the CBA and to show what would happen given certain combinations of assumptions.

Let b_{it} be the value of the ith benefit received in year t, and c_{it} be the value of the ith cost paid in year t. Then the NPV expression is

$$NPV = \sum_t \frac{\sum_i b_{it} - \sum_i C_{it}}{(1+r)^t}$$

Each component benefit (b_{it}) and cost (c_{it}) is often an estimate. Thus the accuracy of the NPV calculation depends on the accuracy of these estimates. There are three approaches to sensitivity analysis that can be used to address the degree of error in these estimates.

Subjective Estimates

Based upon experience and insight, the analyst might state that the NPV is subject to an error of plus or minus 10%. A subjective estimate can be quite good depending on the skill of the analyst. The advantages of a subjective estimate are that it is quick, inexpensive, and can account for variability not reflected in the objective measures. The disadvantages are that it does not have a quantitative basis and the analyst could have difficulty in defending the estimate to critics.

Selective Sensitivity Analysis

The analyst selects parameters involved in the NPV calculation that he believes are subject to error and that could significantly affect the result. For each of these parameters, he selects likely high and low values and computes NPV values with each. The decision maker is then presented with three NPV estimates - high, medium (the original value), and low.

The advantages of selective sensitivity analysis are that it is objective and easy to compute. The disadvantage is that it is only suitable for situations where only a small number of parameters are subject to error.

General Sensitivity Analysis

This approach derives a probability distribution of NPV outcomes. Each b_{it} and c_{it} depend, in general, on a number of parameters. Call these parameters the set $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_K\}$.

Suppose high, medium and low estimates are available for each α_i. Now partition the set of parameters into disjoint subsets A_j such that all parameters are placed in the same set if and only if they are dependent on each other. If two parameters are independent they must be in different subsets. Thus each α_i must be a member of exactly one subset A_j. Since the α_i's in each A_j are related, there are only certain combinations of values each A_j can assume. The analyst must determine each of these combinations and their corresponding probabilities. Suppose the set A_j can assume θ_j configurations. Denote these configu-

rations as $A_{j1}, A_{j2}, \ldots, A_{j\theta_j}$ and the corresponding probabilities as $P(A_{j1}), P(A_{j2}), \ldots, P(A_{j\theta_j})$. The NPV cumulative probability distribution can be computed from these combinations of parameters to provide risk information to the decision maker in a convenient format.

Risk and Uncertainty

If no meaningful probability can be assigned to certain sets of the parameters, then this situation is called an uncertainty, whereas the situation in which probabilities are assignable is called a situation of risk. The discussion above addresses risk. One way to handle uncertainty is to give ranges of estimates for each uncertain parameter. When the results of the CBA are presented, a payoff matrix can be presented which computes the NPV under each assumed value for the parameter.

2.5 Summary

The software engineering literature contains a large amount of information that can be used to help support process improvement decisions including empirical data, process improvement frameworks, cost-models, economic models, and simulation models. As we have seen, these approaches have limitations and no single approach is adequate for evaluating the full impact of an SPI within a specific organization. The cost-benefit analysis and investment analysis literature suggests a systematic method and framework for evaluating

potential SPIs as potential investments. Such a framework can serve to organize the available data and models and provide decision support to practitioners.

Chapter 3

A Framework for Evaluating Software Process Improvement Investments

This chapter develops a general, unifying framework to support evaluating software process improvements on the basis of their economic desirability. SPI proposals can be viewed as potential investment alternatives aimed at improving quality, cost, and schedule of software development. Our goal is to provide a framework that can be used to build an organized repository of information and models for potential process improvements. For each SPI a template must be constructed to identify the set of cost-benefit effects along with quantification functions and parameters based upon the best available industry data or models. The relationships among SPIs should be identified to help the decision maker understand which SPIs are mutually exclusive, which are prerequisites, and which are complementary.

3.1 A Formal Model of Investing in Process Improvement

This section develops a general model of the investment decision for software process improvements. The purpose of any investment analysis is to determine which investment alternative is the best use of the organization's resources.

3.1.1 Cost-Benefit Analysis Principles

Our CBA framework is based on the principles of cost-benefit analysis as stated by technology-economist Peter Sassone [70], highlights of which are:

1. Effects will be expressed in dollars and schedule impacts.

2. Use of discounted cash flow analysis to account for the time value of cost and schedule impacts.

3. Use of life cycle cost-benefit analysis.

4. Adoption of with-without rather than the before-after perspective in comparing alternatives.

5. Use of net present value as the single best financial criterion in aggregating costs and benefits over time.

6. Use of corporate opportunity cost of capital as the appropriate discount rate in discounted cash flow calculations.

Unfortunately, little of the software engineering literature uses these standard principles. The few notable exceptions include Vienneau [76], Cruickshank [18], and Slaughter, et. al. [72].

3.1.2 Net Present Value

Our framework uses the Net Present Value (NPV) criterion for evaluating proposals.

Benefit and Cost Effects

Let $E_t = B_t - C_t$ be the total cost-benefit impact during time period t where B_t is the total value of the benefits received and C_t is the total of costs incurred during time period t. The total cost-benefit impact E_t can be divided into a number of subcategories of cost-benefit effects. Let e_{jt} be the value of the jth cost or benefit effect that is expected to occur during time period t. If $e_{jt} > 0$, the effect will be considered a benefit for that time period, otherwise it will be considered a cost. Then

$$E_t = \sum_j e_{jt} \qquad \text{(EQ 2)}$$

and the NPV equation becomes

$$NPV = \sum_{t=0}^{n} \frac{\sum_{j} e_{jt}}{(1+r)^t} \quad \text{(EQ 3)}$$

We will let NPV_j represent the net present value for a fixed cost or benefit effect j. Thus,

$$NPV_j = \sum_{t=0}^{n} \frac{e_{jt}}{(1+r)^t} \quad \text{(EQ 4)}$$

and

$$\sum_{j} NPV_j = \sum_{j} \sum_{t=0}^{n} \frac{e_{jt}}{(1+r)^t} = \sum_{t=0}^{n} \frac{\sum_{j} e_{jt}}{(1+r)^t} = NPV \quad \text{(EQ 5)}$$

Each cost-benefit effect, e_{jt}, can be estimated by a real valued function f_{jt} whose domain is $X \times X \times \aleph$, where $\aleph$ is the set of natural numbers representing the applicable time period and each X represents a set of vectors for predicting marginal impacts to the software development process. That is, $X = \{\langle x_1, x_2, \ldots, x_M \rangle | x_k \in X_k, 1 \le k \le M\}$ where each X_k represents the domain of possible values for the kth parameter. The first vector represents the estimated parameters for the baseline scenario and the second vector represents the estimated parameters for the alternative scenario under consideration. The

estimation function f can be defined in terms of function g, where g quantifies the cash flow of effect category j for time period t under the given scenario $\boldsymbol{x}$.

Then at each point in time t,

$$f_{jt}(\boldsymbol{x}, \boldsymbol{y}) = g_{jt}(\boldsymbol{y}) - g_{jt}(\boldsymbol{x}) \qquad \text{(EQ 6)}$$

The jth cost or benefit effect at time t is the difference between the net cash flow for effect category j under conditions of an alternative in place ($\boldsymbol{y}$) and the net cash flow under the condition of the baseline ($\boldsymbol{x}$) in effect. That is, f is the monetary difference that occurs for the effect category j and time period t when conditions $\boldsymbol{y}$ are in effect.

We also define $\boldsymbol{f_j}$ and $\boldsymbol{g_j}$ as vector valued functions that return a stream of values for each time period. That is $\boldsymbol{f_j} = \langle f_{j0}, f_{j1}, \ldots, f_{jn} \rangle$ and $\boldsymbol{g_j} = \langle g_{j0}, g_{j1}, \ldots, g_{jn} \rangle$. If we define a discount factor as $d = \frac{1}{1+r}$ and let the vector $\boldsymbol{d} = \langle d^0, d^1, \ldots, d^n \rangle$ the formula for net present value may be written in vector notation. The NPV is a function of the baseline parameters, parameters under an alternative scenario, and a discount factor

$$NPV(\boldsymbol{x}, \boldsymbol{y}, d) = \sum_j NPV_j(\boldsymbol{x}, \boldsymbol{y}, d) = \sum_j (\boldsymbol{f_j}(\boldsymbol{x}, \boldsymbol{y}) \cdot \boldsymbol{d}) = \boldsymbol{d} \cdot \sum_j \boldsymbol{f_j}(\boldsymbol{x}, \boldsymbol{y}) \qquad \text{(EQ 7)}$$

This formulation allows us to consider the stream of values for each effect j separately. An estimation function $\boldsymbol{f_j}$ must be developed for each primary effect j in terms of the baseline parameters and the parameters for the alternative scenario.

3.1.3 Parameter Attributes of a Development Organization

Suppose a manager is considering a set of potential improvements to apply to a baseline environment. The baseline process and environment of an organization can be described by a set of attributes. Examples of attributes would include code size, labor costs, defect rates and productivity rates. Let measures of these attributes be denoted by the vector

$$x = \langle x_1, x_2, \ldots, x_M \rangle \quad \text{(EQ 8)}$$

Note that no particular structure is imposed on a given parameter x_i. Some of these parameters may themselves be vectors or functions indexed by time.

Let $n > 0$ be the maximum number of time periods in the future over which the project will be evaluated. Let the set of positive numbers T be the time interval in the future over which the project will be evaluated and $t \in T, 0 \leq t \leq n$ an index into the time horizon. Each t represents a point in future time with $t = 0$ representing the present. For cost-benefit analysis, time periods are typically years, but could be in any convenient unit of time (months, weeks, days, hours). Each x_k could be indexed by the time.

Let $I_1, I_2, \ldots, I_{N_p}$ denote a set of N_p process improvement proposals under consideration and let I_0 represent the baseline alternative. For each improvement I_j we can estimate values for a parameter vector $x_j \in X_j$ which we believe will be in effect for that alternative. Again, we do not impose any constraints on the structure of the individual parameters. For a given effect category, distinct estimation functions may be required to

estimate the impacts for distinct improvements. Thus, $f_{i,j}$ is a vector valued function that evaluates effect category j for process improvement i.

Let N_e be the number of effect categories that have been identified. (Categorization of effects will be discussed in the next section.) The structure for assessing effect categories across a number of potential improvements is shown in Table 3.1.

Table 3.1: Improvement proposal versus effect category matrix

Improvement \ Effect	e_1 *effect 1*	e_2 *effect 2*	...	e_{N_e} *effect* N_e
I_0 Baseline alternative	$f_{0,1}(x_0, x_0)$	$f_{0,2}(x_0, x_0)$	...	$f_{0,N_e}(x_0, x_0)$
I_1 Improvement 1	$f_{1,1}(x_0, x_1)$	$f_{1,2}(x_0, x_1)$	...	$f_{1,N_e}(x_0, x_1)$
I_2 Improvement 2	$f_{2,1}(x_0, x_2)$	$f_{2,2}(x_0, x_2)$	...	$f_{2,N_e}(x_0, x_2)$
⋮	⋮	⋮		⋮
I_{N_p} Improvement N_p	$f_{N_p,1}(x_0, x_{N_p})$	$f_{N_p,2}(x_0, x_{N_p})$	...	$f_{N_p,N_e}(x_0, x_{N_p})$

Note that the effect functions for the baseline alternative all evaluate to **0** since for any effect i

$$f_{0,i}(x_0, x_0) = g_{0,i}(x_0) - g_{0,i}(x_0) = \mathbf{0}$$

This simple fact emphasizes that the decision structure is based upon a comparison of marginal effects to the baseline. Cash flows that are not affected by an improvement should have no impact on the decision.

3.1.4 Cost-Benefit Effect Classification

There are a wide range of cost-benefit effects that have been claimed for the various software process improvements in the literature. To bring some order to this chaos, we establish a hierarchical classification taxonomy that will serve as an aide in identifying the significant cost-benefit effects for a SPI proposal. This taxonomy also can be used by software managers to help identify improvements to achieve desired benefits.

The top level cost-benefit effect accounting categories for this taxonomy are:

1. Implementation and Support

   The costs of implementing and sustaining the process improvement.

2. Production Effects

   Staff effort cost impacts for developing software products as well as for indirect management and support cost. This category omits defect detection and resulting rework and repair costs.

3. Quality Effects

   The effect the improvement has on quality costs. Quality costs include the costs of assessing quality in software products and the costs that results from poor quality in software products.

4. Cycle Time

   The percent calendar time improvement the SPI is expected to have on the software product cycle time.

5. Customer / Market Effects

   Financial effects based on how the process improvement impacts the product's marketplace.

6. Other Effects

   This is a catch-all category for other effects that do not fit into the first five categories.

The full taxonomy of cost-benefit effects is listed in Appendix A. Effects for a particular improvement will be attached to nodes in this classification tree. Effects will be quantified by functions associated with leaf nodes in the hierarchy. The net present value impacts of cost-benefit effects can be summarized at each interior node in the hierarchy of this classification.

3.1.5 Drivers Behind the Functions

Each function in the taxonomy delivers the data required by the NPV and ROI calculations. All functions have certain parameters in common, but may vary in many other respects.

Common parameters

- SEI CMM level of the organization
- Industry software subcategory classification

- Size of current code inventory (lines of code or function points)
- Percentage of code inventory being changed per year
- Amount of new code to be developed per year
- Current testing and verification process being used
- Number of personnel
- Cost of personnel (loaded labor rate)
- Size of development budget
- Cost of failures (internal and external)

Functions

A variety of approaches can be utilized to estimate the effects of an SPI. Estimating the effects is the most difficult aspect of performing a cost-benefit analysis. The framework provides for estimation functions with suggested parameters for those effects based on the best available industry information. Estimation approaches include:

- Industry data or customer experience
- Mathematical models (simple or complex)
- Dynamic simulation models

3.1.6 Justification of Choices Made

Each choice and decision should be justified as well as possible. As discussed earlier, very few software engineering experiments or case studies are conducted under scientifically acceptable designs and controls. Instead, authors report the facts and conclusions as best they can, but few reports can be taken as more than anecdotal evidence. Still, one data point is better than none. Better choices will emerge over time and the framework will readily accommodate the better choices. Approaches used will include

- Literature reports of correlational studies
- Literature reports of case studies
- Private, internal company data
- Compiled industry data
- Existing economic or simulation modeling approaches
- Analysis based on the role of the function, type of effect
- Expert opinion
- Folklore of the industry

The decision maker can be given rationale for each of the cost-benefit effects along with quality of data, success factors and potential pitfalls of the process improvement under consideration.

3.2 Using the Framework

A systematic procedure for performing economic analysis of process improvement using the framework follows.

1. Recognizing a problem or opportunity

   This is the starting point for an organization to consider process improvement. Often the recognition of the need for process improvement is obvious but only occurs after much damage has already been done. Some combination of missed deadlines, blown budgets, and unacceptable quality will prompt management to seek solutions to these problems. More proactive management will detect more subtle warning indicators of problem areas earlier in the life cycle. Metrics programs and improvement models such as the GQM paradigm can be invaluable to make problems visible before it is too late. Problems or opportunities can be defined in terms of the effects they are experiencing and would like to correct or improve.

2. Identify alternative SPIs

   The CBA-SPI framework helps management identify potential SPI solutions. The effects that they would like to change can be compared to the framework to identify SPIs that address those effects. The relationships among SPIs can suggest which SPIs to consider first.

3. Determine the opportunity cost (discount rate)

   This should be based on the corporate cost of capital and the risk of the benefits to be received in future time periods.

4. Determine the time horizon

   The time horizon will depend on the type of improvement and the environment where it is to be implemented. A 5-10 year horizon is usually sufficient. Since future benefits are discounted, benefits are greatly diminished beyond 5-10 years and would have minimal impact on the decision.

5. For each SPI

   a. Identify the cost-benefit effects

      The framework provides significant assistance to the decision maker by identifying the potential cost-benefit effects an organization may expect to receive from an SPI. A decision analyst may choose to add or subtract from the provided list of potential effects.

b. Estimate and quantify cash flows for each effect

The framework allows the decision maker to estimate cash flows by time periods in the future for each effect. Estimation models and default parameters help facilitate the estimation process, but allow the estimator the flexibility to override values as needed.

c. Calculate the NPV and ROI metrics for each effect and summarize for the SPI

These calculations follow from the effects and their cash flows.

6. Compare alternatives

The summarized cost-benefit information provides support to the decision maker. The decision process of Section 2.4.2, "Structuring the Decision Problem," can be used to help determine the best solution.

3.2.1 Template Construction and Validation

The framework requires that a template be constructed for each SPI consisting of the cost-benefit effects, evaluation functions, required parameters, model justification, prerequisites, and success factors. Template construction and validation are described in general here and illustrated in detail in chapters 5 and 6.

Template Construction

The following procedure was developed to guide the construction of an SPI template.

1. Review effects currently in the framework for similar SPIs

   Identify effects that may apply to the current SPI based on effects already in the database for similar SPIs.

2. Survey literature and experience for the SPI

   Review correlational studies, case studies, current economic models, interviews with practitioners, simulation models, claims of experts, and success factors related to the SPI.

3. Identify the reported cost-benefit effects and determine the most significant effects

   Incorporate these effects into the SPI template and classify the effects by the taxonomy of Appendix A, by the Cost of Software Quality category, and by life-cycle development phase.

4. Analyze the dynamics and causes behind each significant effect.

   Develop models to understand the dynamics that explain the effect (e.g., between possible parameters and resulting cost-schedule impacts).

5. Design and develop models or functions for the significant effects

   The parameters required for the models should be readily available or easy to estimate for most practitioners.

6. Document functions (or models) and assumptions used

7. Define dependencies between the new SPI and other pre-defined SPIs

   When comparing two SPIs, the first SPI could be (a prerequisite of, a complement of, independent of, a substitute for, mutually exclusive with) the second SPI. These relationships should be explicitly defined in the template.

Template Validation

Template validation is the process of building confidence in the soundness and usefulness of an SPI template. The soundness of a template should be judged on the basis of how well it identifies, justifies, and quantifies the cost-benefit effects associated with a potential process improvement. However, the role of a template is not to produce a perfect predictive model of the process improvement. The accuracy of how well a template models the SPI's cost-benefits should be judged relative to the available empirical evidence for the SPI. Note that when an analyst uses a template to construct a CBA, the CBA itself must be validated to ensure the results are reasonable for the intended environment.

The usefulness of an SPI template should be measured by how well it facilitates the process of constructing a business case for the process improvement within a particular environment. The SPI template should reduce the effort required to construct a CBA and result in better economic justification for the time spent.

The process of building confidence in the template involves a combination of structural analysis, peer review, user testing, and comparing model results with available empirical data and economic models. Structural analysis involves verifying that the set of effects in the template include all significant effects known for the SPI, that all cost-bene-

fit effects have reasonable justification, and that effects are not double counted. The results should be self-consistent and reasonable. For example, an estimate of the SPI savings for a time period should not exceed the annual development budget.

Peer review of an SPI template with experts and practitioners of the SPI can provide valuable feedback on how plausible and accurate the SPI template matches their experience and judgement. All aspects of the SPI template are subject to peer review including: the identified cost-benefit effects, the justification for each effect, the estimation functions and required parameters.

User testing involves having software practitioners or SPI experts use the framework to construct a CBA for the SPI. The CBA could be for a hypothetical situation or, preferably, for a real software organization that could potentially benefit from the SPI. This form of testing provides feedback on the ease of use; difficulty of obtaining parameters to drive the models; and the reasonableness, credibility, and strength of the resulting business case.

The most effective form of validation compares the CBA results from the template against known empirical data for the SPI. Empirical data can include published case studies, economic models, or retrospective follow-up CBA studies from organizations that have implemented the SPI.

The feedback from these various forms of validation offers opportunities to continuously adapt, calibrate, and improve the accuracy and usability of the templates. Actual cost-benefits can be compared to those predicted by the template. Differences can be analyzed to understand and document the reasons for the variations. Variations may be caused

by unique conditions associated with the development environment or possibly indicate the need to make modifications to the template.

Chapter 4

Architecture and Design of Prototype

CBA-SPI is a prototype tool to assist software managers in preparing cost-benefit analyses for use in evaluating software process improvement initiatives. Such a cost-benefit analysis can be used for any of the following purposes:

- To determine the potential value of a proposed SPI to support the business case for its implementation.
- To aid in planning allocation of resources for software process improvement.
- To audit the costs and benefits for a previously implemented SPI.
- To assist in pricing an SPI product by estimating the value to be received by potential customers.

CBA-SPI is based upon the framework described in the previous chapter and contains a collection of software engineering data from the literature as well as information about potential process improvements. Appendix B describes the underlying database schema for the tool, and Appendix C provides example user interface forms. This chapter describes the high level functional design and architecture.

4.1 Functional Overview

The CBA-SPI tool set provides six main areas of functionality: 1) define SPI templates, 2) define cost-benefit effects, 3) provide industry data and models, 4) run software process simulations, 5) define baseline environments, and 6) calculate cost-benefit analyses.

4.1.1 Define SPI Templates

The Define SPI Template function is to collect and organize information for a software process improvement. The information requested by this function includes identification and description of the SPI, published ROI data, expected defect removal efficiency impacts, cost-benefit effects, effect formulas, required parameters, success factors, and relationships to other SPIs. A *defect removal efficiency impact* refers to a percentage change in defect removal efficiency that the SPI is expected to have on a QA process. The defect removal efficiency impacts are used by a general model built into the tool which estimates defect removal efficiency and quality failure costs for an organization's process. A cost-benefit effect may have multiple formulas each assigned to different time periods. The user may choose from a list of defined parameters to include in formulas.

A separate option is used to define parameters. A parameter definition includes an identifier, variable name, data type, length, user prompt, explanation, default value, and a valid range. Parameters may also have their own formulas for computing their default values. The set of parameters for all effect functions of an SPI are explicitly linked to the template.

The SPI template may be printed in a report format.

4.1.2 Define Cost-Benefit Effects

The Cost-benefit Effects function allows navigation and modification of the cost-benefit effects taxonomy. The definition of each cost-benefit effect includes the effect identifier, effect name, cost/benefit indicator, effect explanation, tangibility code, effect categories, SPI templates that assign the effect, and links to parent and child effects. Effects can be assigned Balanced Scorecard and Cost of Quality categories.

4.1.3 Provide Industry Data and Models

Industry data and estimation models are provided by the CBA-SPI tool to assist in calibrating the baseline scenario and estimating parameters for the cost-benefit effect functions. The sections below give examples of available data and how it is organized.

Process Step Data

Process related industry information is organized by a process step taxonomy. A process step is a development activity classified as one of the following types: summary, produce, assess, repair, and manage.

Summary process steps are the standard life-cycle development phases (e.g., requirements, design, coding, testing, and maintenance phases). Industry data provided for these

phases includes average defects generated per function point, defect removal efficiency, and defects delivered per function point.

A *produce* step is an activity that directly produces a work product such as source code or documentation. Examples of produce steps include "Specify Requirements" and "Develop Data Model."

Manage activities plan, direct, control, or administratively support the software development effort. An example manage activity is "Develop Release Plans."

An *assess* activity is a QA step. Examples of assess activities include design reviews, functional verification, or system testing. Each QA step has an associated average defect removal efficiency and is linked to an associated repair step.

A *repair* step includes all effort to analyze, repair and retest defects found by the associated QA activity. Three levels of repair effort hours per defect are provided for repair activities as reported by Jones [37].

SEI CMM Level

The prototype provides data from Jones [37] for each SEI CMM level including the average, minimum, and maximum delivered defects per function point, percentage of organizations at the level, defect potential per function point, and the removal efficiency percentage. Higher CMM levels result in higher quality. Descriptions of the Key Process Areas for each CMM Level and the Key Practices of each Key Process Area are available for organizations who wish to estimate their process maturity.

Function Point and Subindustry Data

The prototype also contains a set of data from Jones [37] referenced by the size of the software inventory (in units of function points), by subindustry classification, or both. Jones derives national averages of software productivity and quality based on his analysis of data from some 4,000 software projects. The subindustry categories are: systems, military, MIS, outsource vendor, commercial, and end-user. The software inventory size categories are 0-1, 2-10, 11-100, 101-1,000, 1,001-10,000, and 10,001-100,000 function points. Examples of the available data attributes are listed in Table 4.1.

COCOMO Estimations

CBA-SPI also provides a function for estimating cost, schedule, productivity, and staffing sizes for software projects using the basic COCOMO equations [9]. The produc-

Table 4.1: Example industry data by function points or subindustry

Attribute	Indexed by	Description
Enhance% / yr	Subindustry and FP	The annual percentage of enhancements to the existing base each year.
FP / Staff month	Subindustry and FP	Average productivity for fully tested, documented code.
Defect origin %	Subindustry and process phase	For example, 40% of defects originate from the coding phase for the systems subindustry.
Canceled percentage	Subindustry	Approximately 25% of MIS projects are canceled.
	Subindustry and FP	Approximately 45% of MIS projects in the 1001-10000 FP size range are canceled.
Schedule months	FP	Projects in the 101-1000 FP range take 27 months to complete.
Staff size	Subindustry and FP	Commercial projects in the 101-1000 FP range use about 10 people on the project.

tivity rates, costs, and schedule are available for the project as a whole and by development phase. The COCOMO estimator can accept project sizes in terms of function points as well as in kilo-lines of source code. To assist in converting project sizes between function points and kilo-lines of source code, a list of development languages is provided with conversion factors (in source lines of code per function point) for most commonly used languages.

4.1.4 Run Software Process Model Simulations

A tool named SimRunner is provided to assist in running and calibrating a software process simulation model. A function in CBA-SPI is used to define the name and location of a system dynamics process model along with input parameters, default values, and result variables. SimRunner calibrates the simulation model to a baseline, runs simulations under various input configurations, and retrieves or graphs result variables of interest.

4.1.5 Define Baseline Environments

The Define Baseline Environment function is to identify and define the baseline ("as-is") scenario for a specific software development organization. A quantitative description of the baseline environment is necessary before we can begin to evaluate the impact of proposed improvements.

The information requested by this option includes the organization's CMM level, its software subindustry classification, the size and growth of its code inventory, the number

of development personnel, budgetary information, and a description of the quality assurance (QA) processes used by the organization. A CMM Evaluation function is available to assist in reviewing CMM key process areas and practices for estimating the organization's CMM level if it is not known. QA processes include all verification and testing steps used to assess product quality and detect defects. Information about each QA step is captured to identify quality costs incurred and defects detected during these stages. Also, annual internal and external failure costs are estimated for these stages. When users assign values to these parameters, they can document the source of their data at the same time.

Many organizations lack historical data or metrics to determine productivity rates and quality costs. An estimator dialog and a failure cost model assists the user in estimating defect levels and quality costs based on industry data.

A Baseline Environment report can be produced for verification that lists the supplied parameters and estimates annual costs for internal and external failures. The data collected by the baseline environment provides a broad set of general parameters that are available for evaluating SPI scenarios.

4.1.6 Create a Cost-Benefit Analyses

The Cost-Benefit Analysis function assists a user in conducting a cost-benefit analysis from collecting and organizing the data through preparing appropriate reports. This function collects information about the cost-benefit analysis itself, such as, the time-line for implementing improvements, the discount rate, and the goals and purpose of the CBA. The CBA record is linked to a baseline scenario and to one or more SPIs being considered

for implementation. For each SPI under consideration, the user is provided a default set of effects, estimation functions, and parameters based upon the SPI template. The user can add or delete cost-benefit effects from the template as desired. The user can direct the program to evaluate the functions. The user may override the values computed by the functions.

The results are compiled into a convenient format that includes the following information for each software process improvement:

- Estimated discounted cash flows for each SPI effect and for each time interval under consideration
- The total NPV for the SPI
- Listing of intangible effects
- Parameters used in the calculations
- Success factors for the SPI
- Relationships to other SPIs

4.2 CBA-SPI Prototype Architecture

The CBA-SPI prototype was developed using a combination of Microsoft Access, Visual Basic, Microsoft Excel, and Vensim. The architecture of the prototype implementation of

the tool is shown in Figure 4.1. This selection of tools and architecture is suitable for a stand-alone, single-user, prototype implementation. The development environment within Access allows for rapid implementation of user interface forms and reports using wizards and data bound controls.

Of course, the research prototype is not suitable for distribution and maintenance as a product. Using the CBA-SPI prototype or an individually distributed product would require each user to have licensed copies of all subsystems and proprietary data. Installation and updates to the program, estimation functions, industry data, etc. would require a mechanism for each user. Also, it would be difficult to collect feedback or data from organizations using the tool. The costs and issues associated with product distribution would make it difficult to reach many potential users. Widespread use and evolution of the framework would best be done as a web application.

4.3 Internet Service Concept

Balanced Economic Analysis of Software Technology Investments is an internet service concept to allow software managers to evaluate process improvement technologies. This section describes the purpose and functionality of the service and a software architecture for its implementation.

By making the functionality available on the web, the costs would be lower and the logistics simpler. By reducing these barriers, it will be feasible to make the service available to a very large audience. Any user with access to a web browser would be able to use

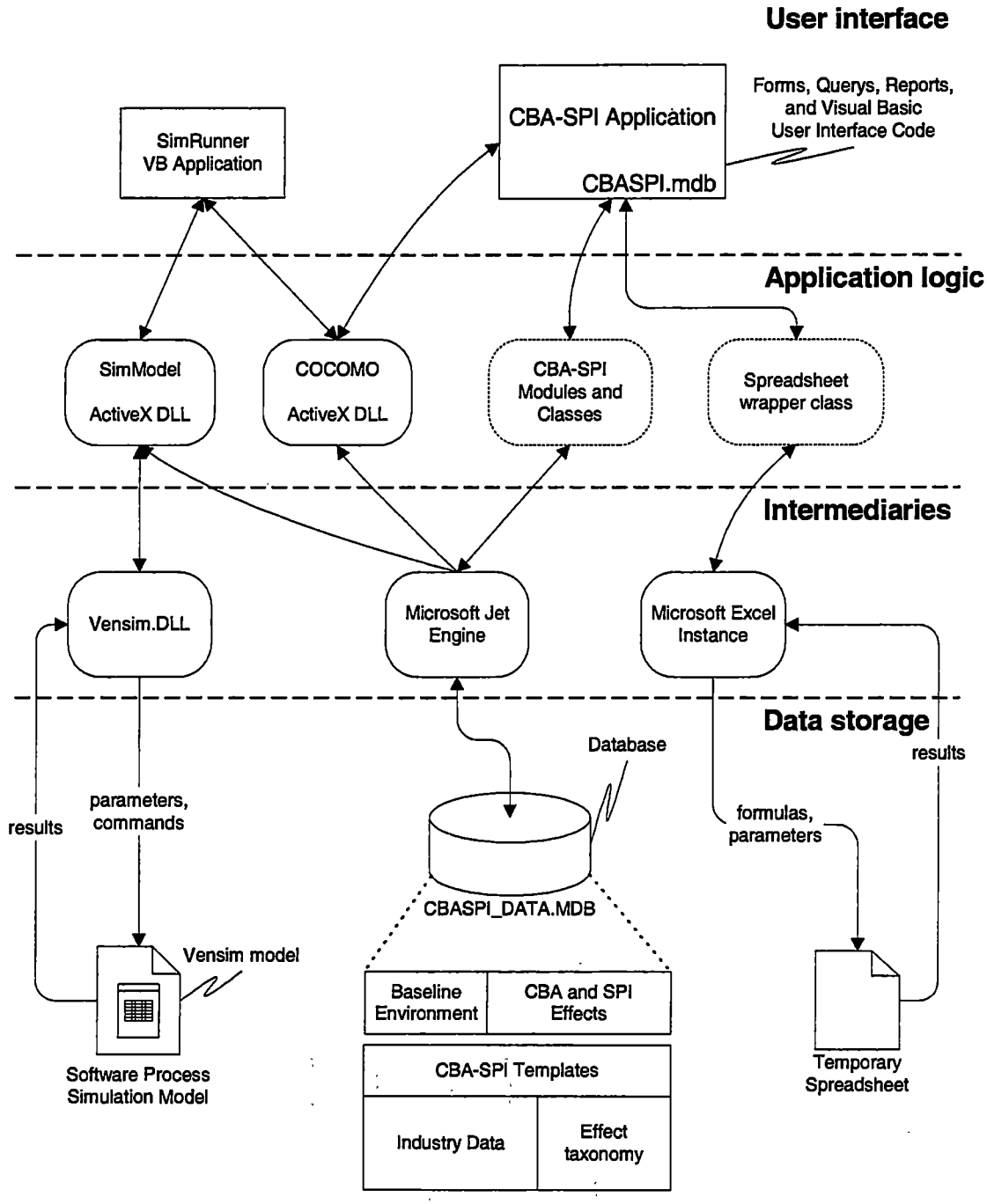


Figure 4.1: CBA-SPI prototype architecture

the service and would not need to spend time installing or maintaining the application software.

By reaching a wide audience at low cost, the service would facilitate transfer of the SPI technologies as well as economic evaluation methods and models for improving software management decision making. Another advantage of a web implementation is it would be much easier to capture feedback on the use of the tool and to collect data from each user's organization. The data collected could be used for continuos improvement of the tool, the SPI templates and models. It could also be used for other data analysis and research purposes. The service would implement data security and privacy policies to protect clients.

4.3.1 User Functionality

The service would provide services for two classes of users: software manager clients and research personnel. The research personnel would be responsible for developing new SPI templates, analyzing data to calibrate and improve existing SPI templates, and updating industry data. The functionality used by research personnel would not be implemented through a web browser and these users would be able to access the full range of functionality that is now in the CBA-SPI prototype.

The clients of the service would be software managers or software engineers who are interested in evaluating SPI for their software development or acquisition. The service would guide these users through the process of establishing a baseline, screening candidate SPIs, and performing full economic evaluation of each alternative. The reports pro-

duced by the service would provide a business case for implementing the SPIs that could be presented to decision makers.

The clients could also use the service for conducting follow-up studies and reporting actual results of previously implemented SPIs. This information would be particularly valuable in improving and calibrating the SPI templates.

Clients would also be able to view industry data, run economic or simulation models, or estimate their process maturity. They would also be able to make changes to their private copy of SPI templates and could create new SPI templates for their own use. However, they would not be able to directly make changes to the base data that is shared by other clients.

4.3.2 Additional Requirements for Web Implementation

Going from a stand-alone, single-user configuration to a multi-user web implementation introduces additional requirements for usability, performance, and security. In the CBA-SPI prototype, the user interface corresponds to the logical organization of the underlying data model. Users must execute several different functions in the process of creating a cost-benefit analysis. The prototype interface has worked well but experimental users have required some training or consultation in order to understand how to use the tool properly. The web implementation must provide a task-oriented interface to walk users through the process step-by-step. In the web environment, clients could be anywhere in the world and will not necessarily have training on how to use the tool or convenient access to the author for consultation.

Because the web implementation would have a shared database with potentially many concurrent users, a database server would be needed to meet higher performance, reliability, and transaction control requirements.

Because multiple users would be interacting with the service and providing sensitive data, the system must provide authentication and data security. A registration service must be provided to allow users access to the system. Users must be able to save their work for retrieval on future sessions. No end-user should be able to view the data of another end-user. Because the data provided by users may be business sensitive, a secure web server protocol would be required between the user and server. Policies and procedures for protecting or using client data must be developed, implemented, and published on the web site.

4.3.3 Architecture

The proposed architecture for the internet service is shown in Figure 4.2.

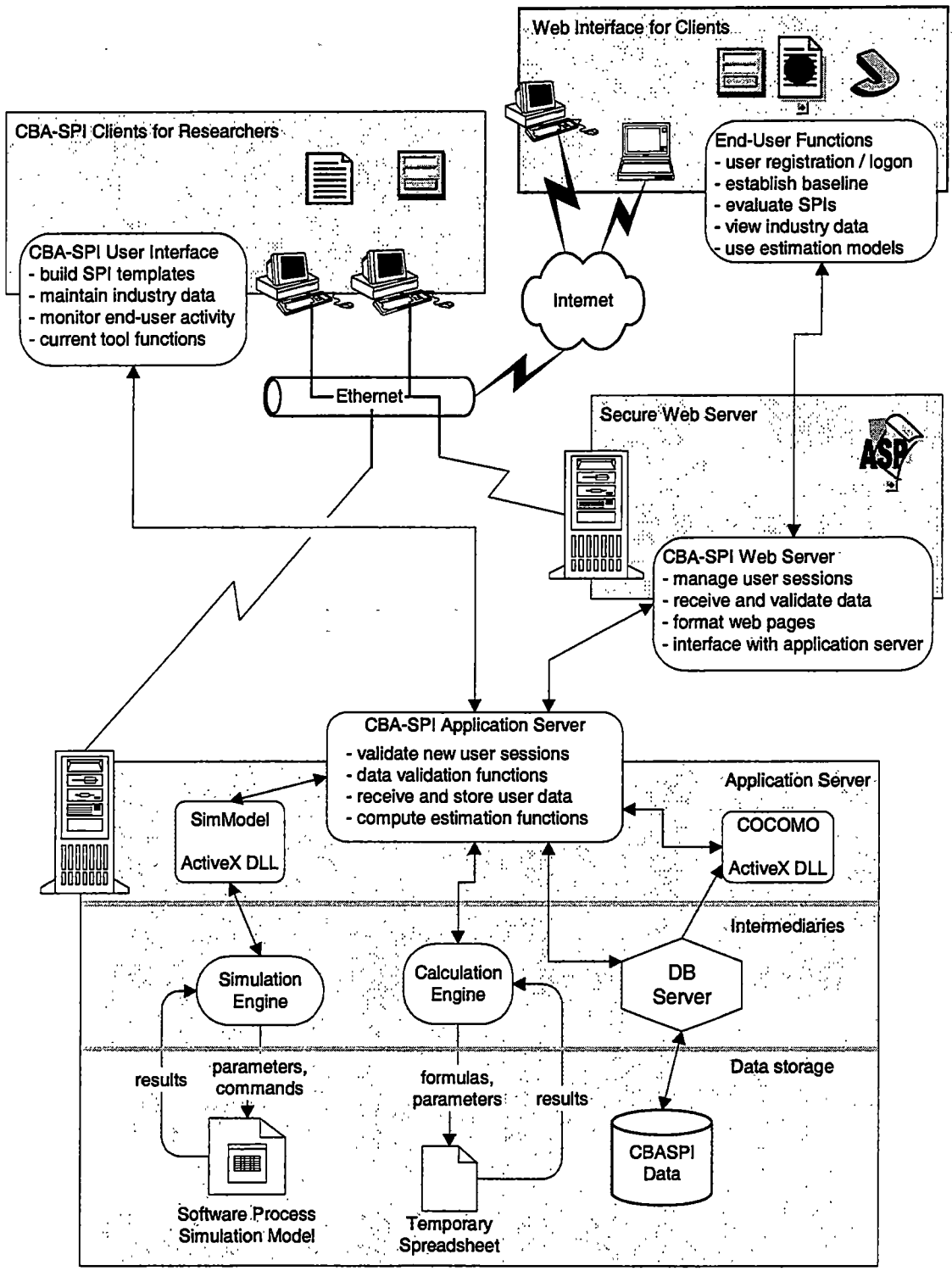


Figure 4.2: Proposed architecture for internet service

Chapter 5

Cost-Benefit Templates for Emerald

Emerald is a software risk assessment product that can analyze the source code of a large software system and predict the high risk areas of the code. This product was developed in 1992 by Nortel Networks as an internal project to improve the reliability of its telephone switch products [35]. Emerald successfully demonstrated its value within Nortel Networks and now is being marketed externally.

Emerald obtains code metrics, process metrics, and use metrics and applies statistical risk models to predict the highest risk areas of the code. These are areas of the code most likely to contain latent defects that escape detection during testing and lead to field failures. Also, these high risk areas are difficult to modify or repair and susceptible to the introduction of new defects.

The information provided by Emerald can be used to improve the software process in a number of ways. In the following three sections, we develop cost-benefit models for three uses of Emerald: targeted defect reduction (Section 5.1), support of reengineering decisions (Section 5.2), and support of the software acquisition process (Section 5.3). For each of these uses of Emerald, we describe how it is used, identify and justify the resulting benefits, and develop models for quantifying those benefits. Section 5.4 defines a cost

model that generally applies to all three of these uses. Section 5.6 summarizes the cost-benefit templates for Emerald.

5.1 Using Emerald for Targeted Defect Reduction

This section examines the use of Emerald for targeted defect reduction. Defect reduction includes activities to prevent defects from occurring as well as activities to find and remove defects. Using Emerald for targeted defect reduction results in a number of benefits. This section identifies these benefits, provides justifications, and develops models for quantifying these benefits for a software development organization.

5.1.1 Overview and Rationale of Benefits

The primary benefits are more efficient resource allocation and a gain from earlier defect removal. Also, secondary benefits of reduced cycle time and improved customer satisfaction can be realized. The diagram in Figure 5.1 illustrates the cause-effect relationships between using Emerald and the potential financial effects.

More Efficient Resource Allocation

Emerald risk metrics can be used to identify the fault prone areas of the code and to predict the number of defects that will occur in the field [35]. By identifying the high risk

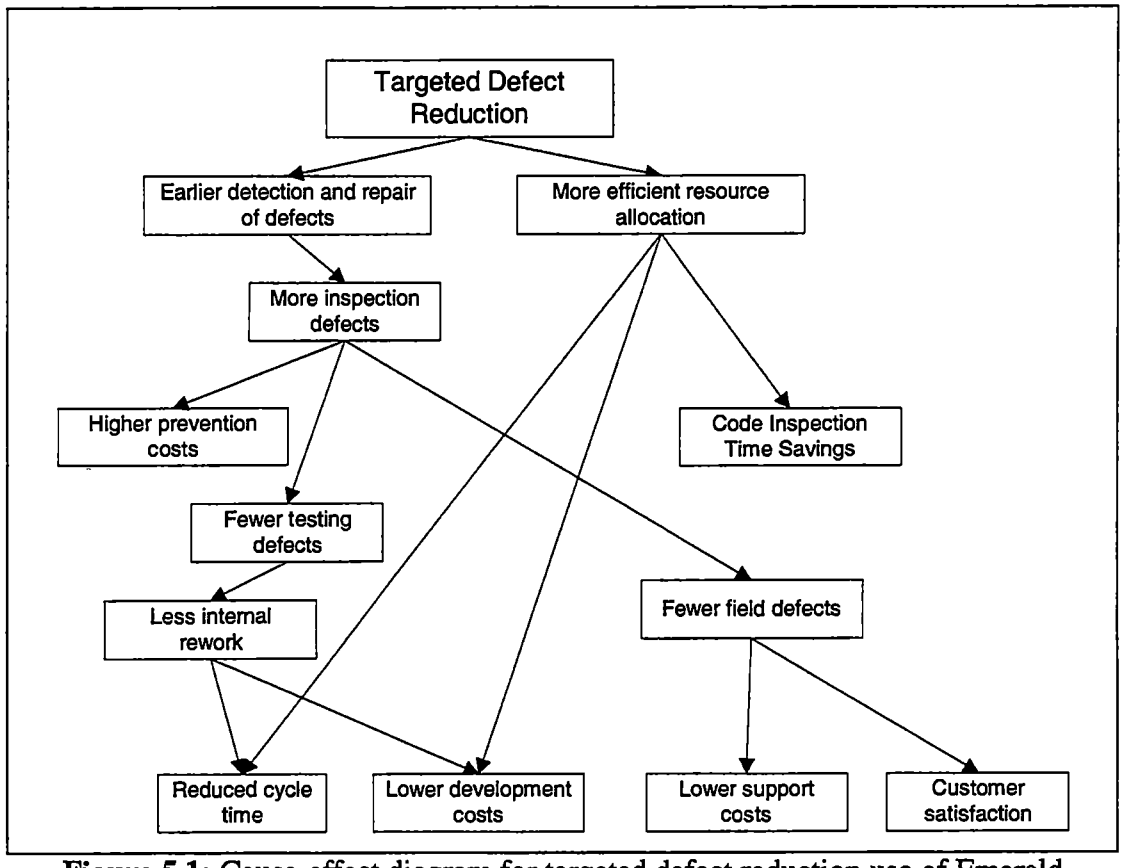


Figure 5.1: Cause-effect diagram for targeted defect reduction use of Emerald

areas in the code, Emerald helps management focus and allocate development, testing and inspection resources where they can most efficiently prevent or remove defects.

Defects in code are not evenly distributed. The 80/20 rule (based on the Pareto Principle) generally applies to changes made to software [3]. That is, 20% of software modules contain 80% of the defects. Stevenson cites several studies to support the contention that most errors and subsequent code changes are concentrated in a small percentage of the modules [73].

Given that the defects are not evenly distributed, it is inefficient to evenly distribute development, testing and inspection resources over all parts of the code. The information from Emerald can help management plan and allocate development, inspection and testing

resources to the appropriate areas of the code. Organizational policies can be established to require special authorization for high risk modules. Management can assign the most qualified developers to modification tasks of high risk code. Inspection and testing resources can be prioritized.

Gain from Early Defect Removal

Defects are less expensive to correct near their point of insertion [41]. The cost of correcting a defect can be 10-150 times more expensive to fix at final testing, delivery and operation stages than during the earlier design and coding stages [68]. Emerald metrics increase the effectiveness of early defect detection efforts. Significant savings are realized from reducing defects that occur in the field and in the final stages of testing. Several Emerald customers report fewer field defects after using Emerald to improve their early defect detection efforts [35].

Secondary Benefits

Important secondary benefits of using Emerald for targeted defect detection include reduced cycle time and improved customer satisfaction.

Reduced Cycle Time. Using Emerald for targeted defect reduction results in higher efficiency of early defect removal efforts and reduces late, pre-release defects. A reduction in rework helps to improve schedule predictability and improve cycle time.

The benefits of reducing cycle time can include: avoiding financial penalties for over-run, receiving financial incentives for meeting targets, greater market share, longer product life, higher profit margins, and freeing resources for other projects.

Improved Customer Satisfaction. Using Emerald for targeted defect reduction reduces errors in the field and promotes on-time product delivery. Reduced field errors and more reliable schedules result in improved customer satisfaction. Customer satisfaction is the vital factor in retaining existing customers and attracting new ones.

5.1.2 Quantifying the Benefits

This section develops models to quantify the cost-benefits from using Emerald.

Quantifying the Benefits from Efficient Resource Allocation

By using Emerald to improve resource allocation an organization can reduce the overall effort required for inspection, testing and development activities. We will illustrate how this effect can be modeled for inspection activities. Similar models can be used for other testing and development activities where Emerald can improve resource allocation. The inspection cost savings model uses the parameters as shown in Table 5.1.

This model assumes that with Emerald the effort will be more concentrated on the high risk ("red") portion of the code with less attention given to the remaining low risk ("green") code. The formula for computing the annual baseline cost for inspecting the code is:

Table 5.1: Baseline parameters for inspection cost savings

Parameter	Description	Example value
reviewed_loc	*Reviewed lines of code.* Number of new or changed lines of code reviewed each year by the organization.	150,000/Year
red_ratio	*Red ratio.* Portion of the reviewed code estimated to be of high risk.	0.2
avg_effort_insp	*Average effort per inspection.*	3.5 hours/inspection
avg_loc_insp	*Average lines of code per baseline inspection.*	250 loc/inspection
cost_staff_hr	*Cost per staff hour.* (fully burdened cost)	$80
grn_effort	*Green effort.* Average effort hours to inspect a low risk ("green") module.	2 hours/inspection
grn_loc_insp	*Green lines of code per inspection.* Average lines of code covered per inspection of a low risk ("green") module.	400 loc/inspection
red_effort	*Red effort.* Average effort hours to inspect a high risk ("red") module.	4 hours/inspection
red_loc_insp	*Red lines of code per inspection.* Average lines of code covered per inspection of a high risk ("red") module.	150 loc/inspection

$$\text{baseline\_cost} = \text{reviewed\_loc} \times \frac{\text{avg\_effort\_insp}}{\text{avg\_loc\_insp}} \times \text{cost\_staff\_hr} \qquad \text{(EQ 9)}$$

The cost for inspecting with Emerald is the expected cost of inspecting both high risk code and low risk code at different levels of effort and concentration:

$$\text{SPI\_insp\_cost} = \text{reviewed\_loc} \times \text{cost\_staff\_hr} \qquad \text{(EQ 10)}$$

$$\times \left(\text{red\_ratio} \times \frac{\text{red\_effort}}{\text{red\_loc\_insp}} + (1 - \text{red\_ratio}) \times \frac{\text{grn\_effort}}{\text{grn\_loc\_insp}} \right)$$

Thus the annual cost savings is

$$cost\_savings = baseline\_cost - SPI\_insp\_cost \quad \text{(EQ 11)}$$

Baseline parameters can be derived or estimated from previous organizational experience. Parameters for estimating the "with Emerald" case should be estimated based on the planned or targeted usage of Emerald within the organization. Table 5.2 provides an example calculation.

Table 5.2: Example of resource allocation cost savings for inspections

Cost per staff hour	$80		
Reviewed LOC per year	120,000		
Baseline			
Avg. effort hours per inspection	3.5		
Avg. LOC per inspection	250		
Effort hours to inspect all code	1680		
Baseline annual cost to inspect code			$134,400
With Emerald	**Red**	**Green**	
Ratio	0.2	0.8	
Lines of code	24,000	96,000	
Effort per inspection	4	2	
LOC per inspection	150	400	
Effort hours per KLOC	27	5	
Effort to inspect	640	480	
Cost to inspect	$51,200	$38,400	
Reliametrics annual cost to inspect code			$89,600
Cost savings			$44,800
Percent savings			33.33%

Quantifying Improved Defect Removal Efficiency

This section develops a model for estimating the cost-benefits of using Emerald to improve the efficiency of defect detection efforts. This model assumes that a certain number of latent defects exist in the code initially. These defects are detected and removed through a series of quality appraisal and repair steps. Late pre-release defects are costly and post-release field defects are particularly expensive.

A series of quality appraisal steps is applied to identify and remove defects. A *quality appraisal step* is an activity to identify potential defects with the code. Examples of quality appraisal activities include: unit testing, inspections, functional verifications, independent testing, beta testing, acceptance testing, and system testing. Jones has published defect removal efficiencies for most common quality appraisal activities [37]. A defect removal efficiency is the number of defects identified by the activity as a percentage of all defects found in the software through the first year of field use.

Each quality appraisal activity has an associated defect repair unit cost to estimate the average cost of correcting a defect found by that activity. Ideally, defect repair costs should be based on an organization's specific experience. However, if this information isn't available default values based on specific sub-industry classifications are available to help development organizations estimate these parameters.

The parameters for this model are described in Table 5.3.

The baseline cost to remove defects at QA step i is:

$$BCost_i = cd_i \times bre_i \times bdef_i$$

Table 5.3: Parameters to estimate benefit from improved defect removal efficiency

Parameter	Description	Example value
defects_yr	*Defects per year.* The number code defects created in new or modified code each year. A separate model is described below to help estimate this parameter.	10,000 defects/yr
The following parameters are associated with the sequence of Quality Appraisal (QA) Steps used by the organization to identify faults (e.g., Informal code inspection, New function testing, Regression testing, System testing, Beta testing). Let n be the number of QA steps, let QA(i) represent the i^{th} QA step and QA(i).<param> be the value of a parameter associated with the i^{th} QA step. The following parameters are required for each QA step.		
cd_i	*Cost per defect.* The average variable repair and failure cost for a defect when it is detected by this QA step. It includes the cost for repairing the defect as well as for other costs resulting from failures generated by the defect. It should not include fixed costs related to defect prevention or testing.	$100 for a defect found during inspection. $1,000 for a defect found during customer acceptance. $10,000 for a field defect.
bre_i	*Baseline removal efficiency.* The baseline removal efficiency of the QA step without making changes to the process. Removal efficiency is defined by the formula $\frac{\text{defects found by QA step}}{\text{defects present at start of QA step}}$	0.5 for code inspection.
ere_i	*SPI removal efficiency.* The removal efficiency you expect to achieve using Emerald. These amounts can be estimated by a percentage improvement over the baseline removal efficiency.	0.575 for code inspection with Emerald. Suggested values are 5-15% over baseline removal efficiencies.

where $bdef_i$ is the estimated number of defects remaining in the product before QA step i is performed using the baseline scenario. We define $bdef_i$ recursively as follows:

$$\begin{aligned} bdef_1 &= \text{defects\_yr} \\ bdef_i &= bdef_{i-1} \times (1 - bre_{i-1}), i > 1 \end{aligned} \quad \text{(EQ 12)}$$

Similarly the annual cost for defects under the Emerald scenario is calculated as

$$ECost_i = cd_i \times ere_i \times edef_i \quad \text{(EQ 13)}$$

where $edef_i$ is the number of defects remaining in the process prior to performing QA step i and is defined as follows:

$$\begin{aligned} edef_1 &= \text{defects\_yr} \\ edef_i &= edef_{i-1} \times (1 - ere_{i-1}), i > 1 \end{aligned} \quad \text{(EQ 14)}$$

The annual savings that can be achieved from this benefit are calculated as the differences of the defect costs under each scenario summed over all QA process steps.

$$Savings = \sum_{i=1}^{n} (BCost_i - ECost_i) \quad \text{(EQ 15)}$$

Estimating defects inserted per year. A key parameter of the above model is defects_yr, the number of defects inserted into new or modified code by the organization over a year. Internal defects from all quality assessment activities as well as field defects

must be included in the count. For organizations who lack this data, this section describes an approach for estimating defects_yr.

Jones has published average defect potentials per function point based upon industrial data for a large number of software projects covering a range of industries [37]. A *function point* is a metric for estimating the size of an application in terms of its observable functionality. He defines *defect potential* as the "total universe of errors or bugs that might be expected in a software project." He gives defect potential per function point values in a table indexed by system size (in terms of function points) and by a subindustry classification as shown in Table 5.4. His data generally indicates that the density of defects increases as the size of the system increases.

Jones has also suggested that the process maturity of an organization can affect its defect potential. He suggests achievable defect potential targets for the five levels of the SEI CMM maturity model as shown in Table 5.5.

Table 5.4: U.S. average defect potentials[a]

FP Size \ subindustry	Systems	Military	MIS	Outsource vendor	Commercial	End-user
<= 1	1.00	1.00	1.00	1.00	1.00	1.00
2 - 10	3.00	3.25	2.00	2.00	2.50	2.50
11-100	5.00	5.50	4.00	3.50	4.00	3.50
101-1000	6.00	6.75	5.00	4.50	5.00	na
1001-10000	7.00	7.50	6.00	5.50	6.00	na
> 10000	8.00	8.50	7.25	6.50	7.50	na

a. Source: [37] Table 3.44, p. 231.

Table 5.5: Defect potentials / defect removal efficiency targets by SEI CMM[a]

SEI CMM Level	Defect potential	Removal efficiency	Delivered defects
CMM 1 Initial	5.00	85%	0.75
CMM 2 Repeatable	4.00	90%	0.40
CMM 3 Defined	3.00	95%	0.15
CMM 4 Managed	2.00	97%	0.06
CMM 5 Optimizing	1.00	99%	0.01

a. Source: [39], Table 4, p. 393.

A case study at Motorola found that the defect injection rate decreased by roughly half each time a project advanced a CMM level [22]. The data in Table 5.5 represents expert opinion and is not directly based on empirical data. However, it provides a general direction and magnitude of the effect that process maturity is expected to have on defect potential and is supported by industrial case studies.

To estimate defects_yr using data from Table 5.4 or Table 5.5, we need the parameters listed in Table 5.6. Our approach for estimating defects per year is to allow a manager to provide the value based on historical organizational data, or to first estimate defects per function point and function points per year and then calculate defects per year as follows:

$$\text{defects\_yr} = \text{defects\_fp} \times \text{fp\_yr} \qquad \text{(EQ 16)}$$

Table 5.6: Parameters for estimating defects per year

Parameter	Definition	Suggested data source
defects_fp	*Defect potential per function Point.* The total number of defects that are injected into the code per function point.	Use historical organizational data if available. Otherwise, estimate using tables of defect potential per function point.
fp_yr	*Function points per year.* Total new or changed function points per year.	Source code library. Two approaches for estimating this parameter are described in the next section.
avg_proj_size	*Average project size.* Average project size in function points.	Total inventory in lines of code divided by number of active projects. Convert results to function points.
subindustry	*Subindustry category.* Select the industry category which best describes the software being produced.	Select one of six subindustries based upon definitions of C. Jones (systems, military, information systems, outsource vendor, commercial, end-user).
SEI_CMM_Lvl	*CMM Level.* Estimated CMM level of the organization.	Formal SEI audit, or self assessment estimate.

Estimating defect potential. Four options are available for estimating defects_fp. The next section provides guidance for estimating fp_yr.

1. Use historical data from the organization to directly estimate defects_fp.

2. Estimate defects_fp by referencing Table 5.4.

3. Estimate defects_fp by referencing Table 5.5.

4. Estimate defects_fp by averaging values from Table 5.4 and Table 5.5.

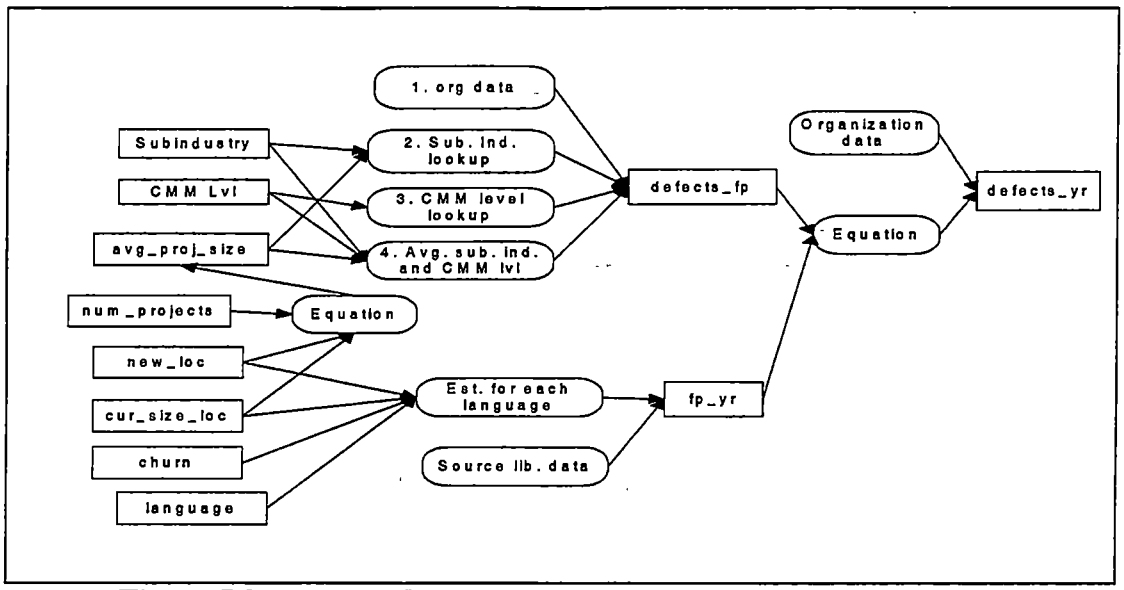


Figure 5.2: Parameter dependency graph to estimate defects per year

5. The potential parameters for estimating defects_yr, defects_fp, fp_yr and their dependencies are shown in the graph of Figure 5.2. The rounded boxes represent estimation functions and the square boxes represent parameters. For some parameters, the user has a choice of several approaches for estimating the function, including an override based on organizational data or a manager's experience. Note that defects_yr is itself a parameter for estimating the savings from using Emerald for targeted defect reduction (EQ 15).

Estimating new or changed function points per year. A key variable in determining defects_yr is fp_yr, a measure in function points of the new or changed software the organization generates in a year. Two approaches for estimating this parameter are:

1. Determine size of current inventory in lines of source code or by function points. Estimate what fraction of the current inventory gets modified each year and esti-

mate new code to be written during the year. Convert these values to function points using the conversion factor for the language. For example, it takes about 128 lines of software written in C to equal one function point. Let lng represent a language used by an organization, and *ltof*(loc,lng) be a function that converts lines of code (loc) to function points for a given language. Then fp_yr is calculated as follows:

$$\text{fp\_yr} = \sum_{\text{lng}} ltof(\text{cur\_size\_loc} \times \text{churn} + \text{new\_loc}, \text{lng}) \qquad \text{(EQ 17)}$$

2. Use tools with the source code library to compare beginning and ending software inventories for the year. Determine the actual new or changed lines of code over a one year period for each language. Convert the total for each language to function points. Sum over all languages used by the organization.

Example. This section provides an example of how to estimate the savings for improving defect removal efficiency as given in (EQ 15). Let us consider a hypothetical XYZ organization that has two major projects with a total inventory of 1,000,000 lines of code written in the C programming language. For the foreseeable future they expect about 10% of this code base to be modified each year. In addition, they expect to develop about 100,000 lines of new C code each year.

By (EQ 17) the new or changed function points per year are estimated as follows:

$$\begin{aligned} \text{fp\_yr} &= ltof(1{,}000{,}000 \times 0.1 + 100{,}000, \text{C}) \\ &= 200{,}000 \div 128 \\ &= 1{,}562.5 \end{aligned}$$

The software XYZ produces falls under the "Systems" subindustry category and the organization has been certified at CMM Level 2. The average project size is

$$\begin{aligned} \text{avg\_proj\_size} &= ltof\left(\frac{\text{cur\_size\_loc} + \text{new\_loc}}{\text{num\_projects}}, \text{lng}\right) \\ &= ltof\left(\frac{1{,}000{,}000 + 100{,}000}{2}, \text{C}\right) \\ &= 550{,}000/128 \\ &= 4{,}297 \end{aligned}$$

By Table 5.4, the defect potential for the subindustry is 7 defects per function point but by Table 5.5 the defect potential for organizations of their maturity level is 4 defects per function point. For our example, we choose to use Table 5.5 and estimate 4 defects per function point. Hence, by (EQ 16), the total defects injected each year by the baseline XYZ organization is

$$\begin{aligned} \text{defects\_yr} &= 1{,}562.5 \times 5.5 \\ &= 8{,}594 \end{aligned}$$

Table 5.7 lists the quality appraisal steps used by the XYZ organization. For each quality appraisal step a default baseline defect removal efficiency is estimated based on industry data published by Jones [37]. The XYZ software manager estimates that by using Emerald he will be able to improve the efficiency of code inspections and system testing

Table 5.7: Example parameters for estimating savings from improved efficiency

	Baseline removal efficiency	SPI Improvement	SPI removal efficiency	Variable cost per defect
Informal design inspections	0.35	5%	0.37	$25
Formal code inspections	0.65	10%	0.72	$50
Regression testing	0.23	5%	0.24	$300
New function testing	0.30	5%	0.32	$300
System testing	0.36	10%	0.40	$800
Customer acceptance testing	0.15	0%	0.15	$1,000
Field Use	1.00	0%	1.00	$5,000

by 10% and improve design inspections, regression testing, and new function testing by 5%.

The software manager derives the variable cost per defect based on historical failure cost data and defect counts for the XYZ organization. We now have all the needed parameters for the model. The result of applying (EQ 15) for the XYZ organization is shown in Table 5.8.

Table 5.8: Example application of equation 15

	Baseline defects detected *(bdef*bre)*	SPI defects detected *(edef*ere)*	Difference	Savings	Cummulative savings
Informal design inspections	3008	3158	-150	($3.760)	($3,760)
Formal code inspections	3631	3886	-256	($12,778)	($16,538)
Regression testing	450	374	76	$22,666	$6,128
New function testing	452	370	81	$24,448	$30,576
System testing	379	319	61	$48,503	$79,079
Customer acceptance testing	101	73	28	$28,241	$107,320
Field Use	573	413	160	$800,159	$907,479
Total	8594	8594		$907,479	

Secondary Benefits

This section develops models for quantifying secondary benefits of using Emerald for targeted defect reduction. Secondary benefits tend to be less tangible and more difficult to quantify than the more direct benefits. However, secondary benefits such as reduced cycle time and increased customer satisfaction can be critical factors for the long run success of an organization.

Reduced cycle time. Emerald reduces schedule because of better utilization of development, inspection and testing resources and by reducing defects and rework from formal testing. For a government contractor, schedule reduction may translate into receiving financial award fees for meeting milestone target dates and avoiding financial penalties. In the business world, reducing time to market can result in increased sales, bigger profit margins, and allowing resources to be applied toward new products to drive future profits.

The parameters for our model of cycle time reduction benefits are given in Table 5.9. The strategy is to estimate the number of project days saved each year (proj_days_saved_yr) and the value of each day of cycle time saved each year (cycle_day_value). Those value are multiplied to yield the annual value of cycle reduction (cycle_reduct_value).

Figure 5.3 illustrates the chain of parameter dependencies. An example calculation is shown in Table 5.10.

The pre-release labor cost savings is the sum of the inspection cost savings from Table 5.2 and the cumulative savings through 'Customer acceptance testing' as shown in Table

Table 5.9: Parameters for estimating the value of cycle reduction

Parameter	Definition	Suggested data source
pre-release_savings	*Pre-release labor cost savings.* The total annual labor cost savings (prior to release) from more efficient allocation of resources and from early defect detection.	Sum the cost savings from more efficient development, inspection and testing resource allocation. Add in the cumulative savings prior to field use from early defect detection.
cost_staff_hr	*Cost per staff hour.* Average cost of one developer hour.	Company data or industry salary surveys for similar organizations.
staff_hrs_saved_yr	*Staff hours saved per year.* Annual staff hour savings from targeted defect reduction.	$\frac{\text{pre-release\_savings}}{\text{cost\_staff\_hr}}$
staff_proj_hrs_day	*Staff hours per day on project.* The average amount of time a staff person spends on project activities.	6 hours/day. Adjust depending on the amount of schedule pressure that exists.
staff_days_saved_yr	*Staff days saved per year.* The average number of staff days saved per year from targeted defect reduction.	$\frac{\text{staff\_hrs\_saved\_yr}}{\text{staff\_proj\_hrs\_day}}$
staff_day_to_proj_day	*Staff day to project day.* The number of staff days required to reduce the cycle time of the project by one day.	Average number of developers per project (e.g., 5 developers per project).
proj_days_saved_yr	*Project days saved per year.* The number of project days saved each year.	$\frac{\text{staff\_days\_saved\_yr}}{\text{staff\_day\_to\_proj\_day}}$
work_days_yr	*Work days per year.* The number of days an employee works each year.	52 weeks times 5 days per week minus holidays and vacation. E.g., work_days_yr = 234 allows for 11 holidays and 3 weeks of vacation.
cycle_reduct_yr	*Cycle time reduction per year.* The ratio of project days saved each year.	$\frac{\text{proj\_days\_saved\_yr}}{\text{work\_days\_yr}}$
cycle_day_value	*Value for each cycle day reduced.* The present value of saving a cycle day in terms of increased sales, awards attained, penalties avoided.	Value depends on the software product and its marketplace.
cycle_reduct_value	*Annual value of cycle reduction.* The present value of cycle reduction accrued for the current year.	proj_days_saved_yr × cycle_day_value

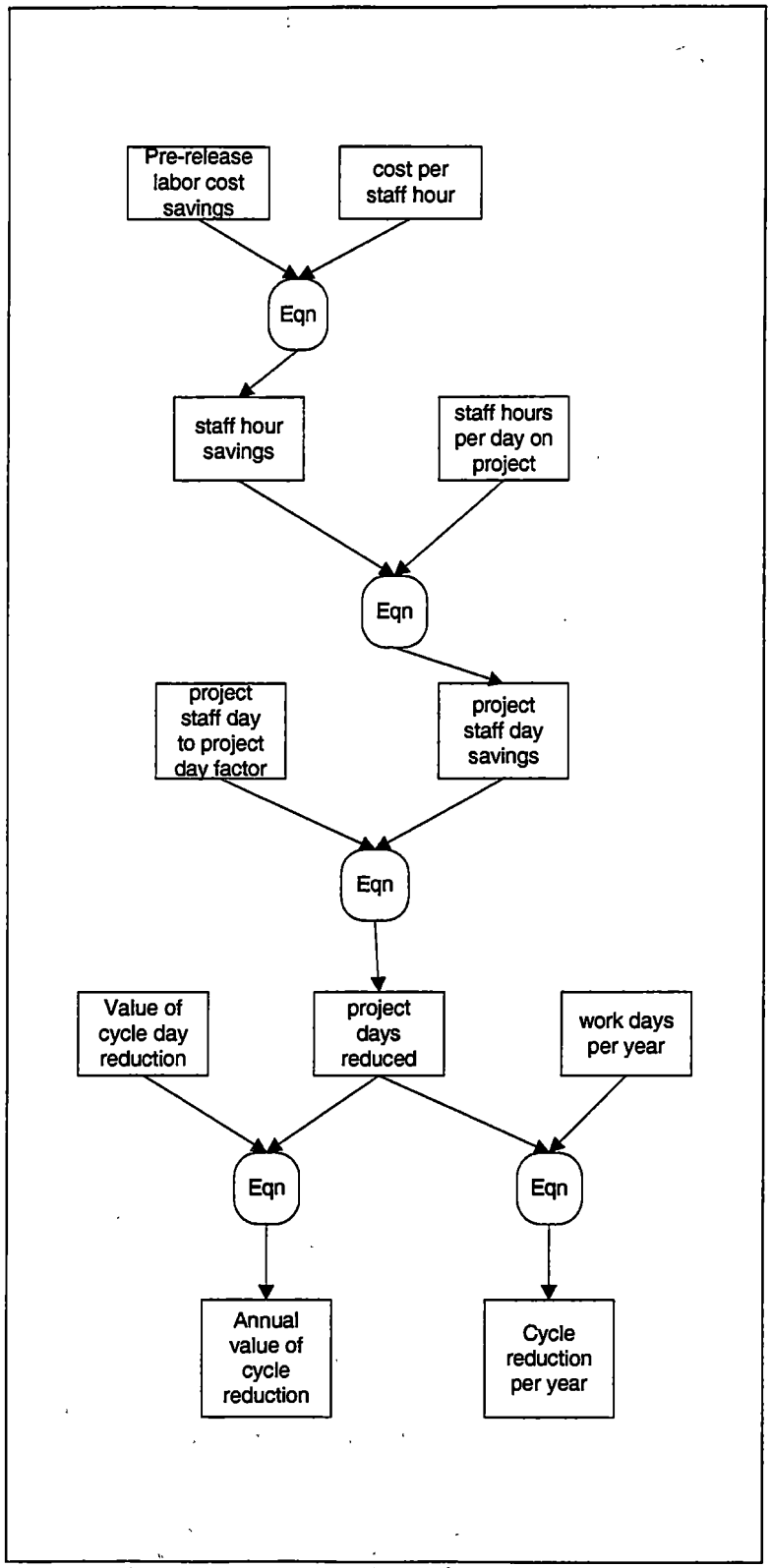


Figure 5.3: Parameter dependency chain for cycle reduction

Table 5.10: Example calculation of cycle reduction value[a]

Inspection cost savings	$44,800
Pre-release labor cost savings	$152,120
Cost per staff hour	$80
Staff hours saved per year	1901
Staff hours per day on project	4.8
Staff days saved per year	396
Staff day to project day	5
Project days saved per year	79
Work days per year	234
Cycle time reduction per year %	34%
Value for each cycle day reduced	$5,000
Annual value of cycle reduction	$396,146

a. Default value for Staff hours per day from [1]

5.8. Note that we could have included savings from improved allocation of testing and development resources had we elaborated those models.

Improved customer satisfaction. Customer satisfaction is frequently mentioned as a benefit of process improvement. For example, Diaz and Sligo stated that Motorola customers value improvements in quality, cycle time and productivity [22]. Many organizations fail to quantify such intangible benefits as customer satisfaction or employee satisfaction. A common practice is to acknowledge that they exist and then ignore them in the calculations. Oxenfeldt recommends that intangibles "must be valued at some specific monetary figure, even while recognizing that such a figure is likely to be incorrect" [59]. McGibbon addresses this issue by providing models for estimating the value of customer satisfaction from process improvement in terms of customer retention and the value of employee satisfaction by reduced employee turnover costs [56].

Table 5.11: Example for estimating an increase in business

	Baseline	w/Emerald	
Repeat business	$1,000,000	$5,000,000	
Additional business		$4,000,000	
Annual net value			$8,000,000

As we have shown earlier, the use of Emerald improves the productivity, cycle time, and quality of the software product. These are characteristics that customers value and that helps to earn their repeat business and attract new customers. McGibbon suggests a straight forward approach as shown in Table 5.11.

To determine the value of repeat business a software manger should

1. consider the projected improvements to cycle time, quality and productivity from using Emerald, and

2. estimate how much repeat or additional business can be achieved from these improvements.

A second approach that we suggest is to relate the quality, productivity, and cycle time improvements to a percentage increase in annual net sales (attributed to repeat business and additional business).

5.2 Emerald's Support of Reengineering Decisions

This section examines the economic value using Emerald to support decisions to reengineer existing code modules. We develop an economic model and provide examples to

show how these benefits can be estimated. A typical scenario for this decision is as follows:

For a new system to be developed, assume that the functional requirements have been specified, the preliminary high level design has been developed, and the system will be composed of a number of components to be developed. Furthermore, for some subset of these components, similar components from a previous development effort have been identified on which to base the new code in the current effort. These existing code units have uncertain quality, maintainability, adaptability and conformance to the current requirements. Most of the existing code components will require at least some adaptations to meet the functional requirements of the new system.

For each component in this subset, the developers are faced with the decision of adaptive reuse versus reengineering. We define *adaptive reuse* as adapting an existing software unit for use in a new application such that needed functional modifications are inserted with the least perceived short-term effort. We use the term *reengineering* to mean the systematic restructuring of an existing software unit in order to improve the software in terms of functionality, performance, reliability, maintainability, reusability and adaptability. Thus, the reengineering choice implies going beyond the functional requirements and improving the quality and long-term costs and schedules associated with maintaining the resulting code. Adaptive reuse on the other hand implies the least effort to get the function to work in the new system, and not addressing any other quality issues that may exist with the code.

Without a tool such as Emerald to assess the risk of the existing modules, we believe that developers would always select the *adaptive reuse* option.[1] When long-term benefits are omitted from consideration, the decision typically is made for the choice with the lowest perceived short-term costs. Since adaptive reuse is perceived to require less up-front effort than reengineering, adaptive reuse would win usually if considering only short-term costs and benefits.

With Emerald, the developer has a better understanding of the risks involved in reusing the existing modules. Long-term benefits are factored into each decision leading more often to correct decisions. For example, if Emerald reveals that a code unit has a history of frequent use, extensive modifications, and high risk of field failures then the choice for reengineering is indicated. Reengineering by definition would improve the quality and cost of future maintenance and enhancements while reducing the risk of field failures and schedule overruns.

5.2.1 Modeling the Benefits of the Improved Decision

Our goal is to develop a model to quantify the value Emerald adds to the decision to reengineer or reuse software modules. For inputs to this model, we need to estimate the average costs to adapt a module and to reengineering a module. We will also need to estimate long-term costs for maintenance of high quality and low quality modules. In addition, we need to estimate what percentage of the time the most economical decision would

1. This assumption is not critical to our model and, in fact, we also model the case that even without Emerald the developer could decide to reengineer.

be made both with and without Emerald. Once we have these inputs we can construct a Bayesian decision model to estimate the benefits. Section 5.2.2 develops an approach to estimating adaptation and reengineering costs. Section 5.2.3 shows how to estimate for long-term maintenance costs for high quality versus low quality code. Section 5.2.4 constructs the decision matrices and provides example calculations.

5.2.2 Estimating Development, Adaptation and Reengineering Costs

Ideally, a company could use financial data from previous projects to estimate its cost of developing new code. Otherwise, industry cost estimates could be used. Jones provides average costs per function point to develop new code for a given subindustry and system size [37]. For example, for a medium sized project (say around 500 function points or 64 KLOCs of C code) in the 'systems' software subindustry we might derive an estimated unit cost of $2,500 per function point. To calculate the average cost of developing a new module, the organization would need to estimate the average size of each module. For example, suppose the average size of a component is 2,048 lines of C code or 16 function points. Then the average cost of a module is 16 times $2,500 or $40,000. This approach will provide us with an estimate of developing *new* code but what we really need is the cost of *adaptive reuse* and the cost of *reengineering* existing code.

We believe reengineering requires more effort than standard *new* development because: 1) the need to understand what the current module is doing requires detailed analysis and reverse engineering to extract precise specifications, and 2) the need for more careful analysis, design, documentation and peer review to ensure the new module does in

fact achieve intended quality, performance and adaptability improvements over its predecessor. In the absence of company data to estimate reengineering costs, we can utilize industry data on software reuse to derive an estimate of reengineering.

Poulin examines a number of studies of the relative cost of writing software to be later reused by others [63]. Writing software to be reused by others requires more effort for additional generalization, documentation, testing and for needed library support and maintenance. He found that the median cost of writing software for subsequent reuse requires about 50% additional effort over new development. The extra effort required to write software for reuse is similar to the extra effort required to reengineer existing code to meet higher standards. So in the absence of empirical evidence to the contrary, we can estimate that the relative cost of reengineering existing software is 1.5, the same as Poulin's estimate for the relative cost of writing for reuse.

Adaptive reuse can be considered a combination of white box reuse and black box reuse. Black box reuse means that no modification is required of the component whereas white box reuse implies that modification will be required. Poulin reviewed the literature for the cost of reusing code and found that the relative cost of black box reuse (compared to new development) ranged from 0.03 to 0.4 with a median of about 0.2 [63].

Poulin does not explicitly calculate a relative cost of white box reuse which requires code modifications to meet functional requirements. However, he cites a study by Selby who reported that reuse with only slight modifications (less than 25% of the code changed) required about 40% the cost of new development. If more than 25% of the code required modifications then the amount of effort increased to 90% of the cost of new

development. Using these default values for relative cost, we can estimate the relative cost of white box reuse as an average that would range between 0.4 and 0.9 depending on the organization. We estimate a default industry value for the relative cost of reuse by assuming that of the reused components that require modification, half require only slight modifications and the other half require more extensive changes. Thus the default value for the relative cost for white box modifications is $0.4 \times 0.5 + 0.9 \times 0.5 = 0.65$.

The relative cost of adaptive reuse for an organization can be estimated as the expected value of the cost of black box and white box reuse. The following is an example calculation for an organization that estimates that 75% of the modules to be reused require some functional adaptations in order to work with the new system.

$$\text{Relative cost of adaptive reuse} = 0.2 \times 0.25 + 0.65 \times 0.75 = 0.5375$$

We can now calculate the estimated cost of reengineering and reuse based on these relative cost estimates and the average size of a code unit considered for reuse. Table 5.12 shows an example calculation of the cost of adaptation and the cost of reuse.

Table 5.12: Example estimation of cost to reengineer and cost of reuse

Language - SLOC/FP	C	128
Avg. SLOC per Code Unit in C	2048	
Avg. Size of Code Unit in FPs	16	
Avg. Cost per FP	$2,500	
Avg. Cost to dev. New Code unit	$40,000	
	relative cost	**cost**
Cost to reengineer	1.5	$60,000
Cost for black box reuse	0.2	$8,000
Cost for white box reuse	0.65	$26,000
Percent of units needing adaptations	0.75	
Cost for reuse	0.5375	$21,500

The main results we need for subsequent calculations are the cost of reengineering a module and the cost for adaptive reuse. Of course, these cost estimates should be calibrated to an organization's past experience if that data is available.

5.2.3 Estimating Enhancement and Maintenance Costs

Stevenson reviewed several studies that were conducted between 1975 and 1985 on the cost of maintenance [73]. He concluded that the proportion of time spent on maintenance can vary widely, from about 30% to 90% of the total cost of the system, but most estimates are from 50% to 70%. These estimates may be low for the 1990's as maintenance costs have been expected to increase over time as a proportion of total development. In 1973 Boehm predicted the relative cost of maintenance would rise to over 80% by 1985 [8] and the conventional wisdom is that this prediction has been fulfilled [76]. With these considerations, we will estimate a present day value for the cost of maintenance to be 72% of the total cost of new development.

Vienneau has pointed out that estimates of the cost of maintenance do not properly discount for the cost of capital [76]. Vienneau derives the ratio of the present value of the cost of maintenance to the present value of the total life-cycle cost as

$$\frac{PV_M}{PV_{TC}} = \left[1 + \frac{(1-p)}{p}\frac{d}{m}(1+r)^d \frac{(1+r)^m - 1}{(1+r)^d - 1}\right]^{-1}$$

where PV_m is the present value of the cost of maintenance, PV_{TC} is the present value of the total life-cycle cost, p is the undiscounted ratio of the cost of maintenance to the total life-cycle cost, d is the number of years in development, m is the number of years in operations and maintenance phase, and r is the discount rate.

Stevenson cites a 1985 study by Fairley that reports the typical life span for a software product is 1-3 years in development, and 5-15 years in use (maintenance) [73]. If we assume the undiscounted cost of maintenance is 72% of the total cost, the discount rate is 10%, the project is in development for 2 years and in operation and maintenance for 10 years, the present value cost of maintenance computes to 60% (using Vienneau's formula) of the present value of the total life-cycle cost. Thus the present value average cost of maintenance is 60/40 = 1.5 times the cost of new development.

Cost of Maintaining High and Low Quality Modules

The purpose of this section is to calculate M_{HQ}, the average cost of maintaining a high quality module, and M_{LQ}, the average cost of maintaining a low quality module. These values will be used later in Section 5.2.4 to estimate the values of the decision payoff matrix. As discussed in Section 5.1.1, the Pareto rule generally applies to software maintenance as most errors and subsequent code changes are concentrated in a small percentage of the modules. The total cost of maintenance (TM) is the sum of the maintenance cost for high quality (TM_{HQ}) and low quality programs (TM_{LQ}). That is,

$$TM_{HQ} + TM_{LQ} = TM \quad \text{(EQ 18)}$$

Let p be the ratio of the total maintenance costs absorbed by the low quality programs then,

$$\begin{aligned} TM_{LQ} &= p \times TM \\ TM_{HQ} &= (1-p) \times TM \end{aligned} \quad \text{(EQ 19)}$$

The cost of maintenance per module, M, can be calculated by dividing the total cost of maintenance by the number of modules, n, in the system, that is $M = TM/n$. If we let q be the ratio of low quality modules to the total number of modules, the cost of maintenance per low quality module, M_{LQ}, is

$$M_{LQ} = \frac{TM_{LQ}}{q \times n} = \frac{p \times TM}{q \times n} = \frac{p}{q} \times M \quad \text{(EQ 20)}$$

and the cost of maintenance per high quality module, M_{HQ}, is

$$M_{HQ} = \frac{TM_{HQ}}{(1-q) \times n} = \frac{(1-p) \times TM}{(1-q) \times n} = \frac{(1-p)}{(1-q)} \times M \quad \text{(EQ 21)}$$

Hence, if we assume that 20% of the code is low quality ($q = 0.2$) and consumes 80% of the maintenance cost ($p = 0.8$) and we estimate the relative cost of code maintenance to be 1.5, then the relative cost of maintaining low quality code is $1.5 \times 4 = 6$, and the relative cost of maintaining high quality code is $1.5/4 = 0.375$. As an example, if the aver-

Table 5.13: Example estimation of cost of maintenance

	relative cost	cost
Maintenance Cost/Module	1.5	$60,000
Maintenance Cost/Low Quality Mod.	6	$240,000
Maintenance Cost/High Quality Mod.	0.375	$15,000

age cost for developing a new module is $40,000 the resulting costs for maintenance is as shown in Table 5.13.

5.2.4 Estimating Decision Results and Probabilities

In the decision we are modeling, we have two choices. We can choose to either reuse the module with possible adaptations or to reengineer the module. In addition, there are two possibilities for the true state of nature of the code module. If the module is not reengineered, it could require either high or low maintenance costs in the future. However, at the time of the decision, the true state of the code module is unknown. With two choices and two unknown states of nature, we have four possibilities that we can examine. We can either:

1. Reuse a Low Quality module,
2. Reuse a High Quality module,
3. Reengineer a Low Quality module, or
4. Reengineer a High Quality module.

In constructing a bayesian decision model, it will be convenient to use matrices for the decision results and probability matrices. Each cell in a matrix will represent one of these four decision possibilities in terms of its cost or its probability of occurrence either with Emerald or without Emerald.

In the following subsections we will consider these possibilities in terms of their potential payoffs or decision result, and how to calculate their probabilities of occurrence both without Emerald, and with Emerald.

Decision Results

A decision result is the cost that will result from a combination of a decision choice and the true state of a software module. There are two simplifying assumptions that we are making about our decision: 1) reengineering will convert a low quality module into a high quality module, and 2) adaptive reuse will not affect the state of a module. These assumptions are supported by our definitions of the decision choices.

Let A be the cost of reuse of a module and let R be the cost of Reengineering a module. Then the resulting cost of each decision possibility can be directly calculated as shown in Table 5.14:

Table 5.14: Cost of each possible decision result

choice \ state	low quality	high quality
reengineer	$R + M_{HQ}$	$R + M_{HQ}$
reuse	$A + M_{LQ}$	$A + M_{HQ}$

Table 5.15: Example decision result

Resulting Costs	Low Quality	High Quality
Reengineer	$75,000	$75,000
Adaptive Reuse	$261,500	$36,500

Table 5.15 gives an example decision result matrix based on the adaptive reuse, reengineering and maintenance cost figures we computed in previous examples.

Estimating baseline decision probabilities

As discussed previously, we assume that in the baseline environment (that is, without the aid of Emerald), the developer would never choose to reengineer. Since the model does not depend on this assumption we will also show how to compute probabilities for the case where the developer could choose to reengineer.

Never reengineer case. Since in this case you can never reengineer, the probability matrix is entirely determined by the estimated percentage of low quality (i.e., high maintenance) modules. Let P_{LQ} be the estimated percentage of low quality modules. The calculation is given in Table 5.16.

Table 5.16: Baseline probabilities for 'never reengineer' case

choice \ state	low quality	high quality
reengineer	0%	0%
reuse	P_{LQ}	$1 - P_{LQ}$

The costs for this scenario is obtained by multiplying the corresponding cells of the probability matrix with the decision result cost matrix and summing. If we assume that 20% of the modules are low quality and use the decision matrix in Table 5.15, the baseline cost per decision is $81,500.

Could reengineer case. We can also assume that even without Emerald, the developer could choose to reengineer. One approach would be to directly estimate all four cells of the probability matrix. However, a simpler method is to estimate the percentage of time that the developer would make the best economical choice. Let B represent the fraction of the time the developer correctly determines (without Emerald) whether a module is low or high quality. That is, B is the fraction of the time the developers choose to reengineer low quality modules and apply adaptive reuse for high quality modules. Then the baseline probability matrix can be calculated as shown in Table 5.17.

If we estimate that $B = 0.75$, $P_{LQ} = 0.20$ and using the example decision result matrix in Table 5.15 our baseline cost per decision evaluates to $64,630.

Table 5.17: Baseline probabilities for 'could reengineer' case

choice \ state	low quality	high quality
reengineer	$B \times P_{LQ}$	$(1-B) \times (1-P_{LQ})$
adaptive reuse	$(1-B) \times P_{LQ}$	$B \times (1-P_{LQ})$

Estimating the decision probabilities when using Emerald

Let E represent the probability that the developer accurately predict the quality of the module and makes the correct choice when using Emerald. Our assumptions are that Emerald will add information to help improve the prediction accuracy of a low quality module. Thus we expect $E > B$. The decision probabilities for each decision are given in Table 5.18.

Table 5.18: Decision probabilities when using Emerald

choice \ state	low quality	high quality
reengineer	$E \times P_{LQ}$	$(1-E) \times (1-P_{LQ})$
adaptive reuse	$(1-E) \times P_{LQ}$	$E \times (1-P_{LQ})$

If we estimate that $E = 0.85$, $P_{LQ} = 0.20$ and use the decision result matrix in Table 5.15 our cost per decision when using Emerald evaluates to $54,415. This would represent a savings of $27,085 per decision when compared to the 'no reengineer' baseline example, or a savings of $10,215 per decision when compared to the 'could Reengineer' baseline example.

Final computations

After arriving at an estimated cost savings per decision, we can annualize this cost by estimating the number of decisions of this nature that would be made during a year. Finally, by providing a discount rate for the cost of capital and an assumed time horizon

for the decision, we can calculate the net present value for the improved value of the decision.

If we estimate 30 decisions per year, a discount rate of 10%, a time horizon of 5 years and a cost savings per decision of $27,085, our annual cost savings is $812,550 and the Net Present Value of these savings over five years is $3,388,224.

5.3 Using Emerald for Software Acquisition

Organizations are often responsible for acquiring custom software from other sources. Outsourcing, a common form of software acquisition, has been a growing trend over the last several years. Outsourcing is associated with a downsizing strategy that many organizations are pursuing. In order to focus on their core business, many software applications are outsourced to third party software development contractors. Another form of software acquisition is associated with a planned company merger or company acquisition where significant software assets will be combined in a larger company. This situation could involve combining software assets in a commercial product line for anticipated competitive benefits. Also, a company merger may involve consolidating various internal accounting, personnel, management information systems and databases.

In any context, the software acquisition process poses many significant challenges and risks for the acquiring organization (i.e., the buyer). Software products are complex and difficult to evaluate *a priori*. Software procurement typically involves the specification and development of code or the code may already exist but require customized modifica-

tions to meet specific requirements. The way in which the acquisition process is managed significantly affects the schedule, quality, costs, risks, and resulting value of the software product to the acquiring organization. This section describes the risks involved in software acquisition and reviews the life-cycle of the acquisition process. We show how Emerald can be used to manage risks through all phases of the acquisition. Finally, we develop models to quantify the benefits an organization can expect to receive from this use of Emerald.

5.3.1 Risks in Software Acquisition

The procurement of software products entails risks. Serious problems exist in acquiring software that is delivered on time, within budget and of acceptable quality. Consider these statistics:

- 30% of outsourcing agreements involve dissatisfaction, a dissolution of the agreement, or some form of litigation within two years [40]
- 28% of all IT (Information Technology) projects fail (1998 Standish Group report).
- 50% of software development schedules are not met [19].
- 10% of software development schedules slip more than 25% of the original estimate [37].

- 33% of software development effort is spent on defect removal and rework [37].

In addition, software is often delivered with many significant defects. The cost of failures in the field can be staggering for both the supplier and the customers of the software. Here are just a few examples:

- A one-day system failure can cost about $5.3-million in lost revenue for an internet auction site such as eBay (Cahners In-Stat Group).

- The downtime for an on-line retailer can cost as much as $10,000 a minute in lost sales (IBM).

- General Motors recalled about 2.5 million pickups, sport utilities and vans with two-wheel drive to correct software on the anti-lock braking systems (AP, Strong, 1999).

- Toshiba Corporation recently agreed to a settle $2.1 billion class action lawsuit because of an error in a floppy-disk controller that could lead to data corruption.

- A defect in the navigation software led to the $125 million Mars Climate Orbiter spacecraft crash in 1999.

There are wide ranges of productivity and quality in organizations that produce software. Rubin reported that the range of performance of software organizations varied by a factor of 600 in 1995. Also, the range in quality of delivered code (measured as post-delivery defects per thousand lines of code) from these organizations varied by a factor of 100

according to Rubin [68]. These wide variations in productivity and quality suggest tremendous benefits that can be generated by improving risk management in the software acquisition process.

There are a variety of potential problems that can occur during the software acquisition process:

- the selected software development company will be incompetent or incapable of developing software of acceptable quality that meets all requirements;
- the acquiring organization will be unable to properly track, monitor and control the project;
- proposed new or changed requirements will fail to be properly evaluated and managed resulting in features that add disproportionately more cost and schedule delays but provide little benefit when put into use;
- the software will be delivered late and over budget;
- the delivered software will not meet requirements;
- the delivered software will be very difficult and expensive to maintain and support;
- the delivered software will be difficult to adapt and evolve to meet changing business requirements.

The acquiring organization can exercise a great deal of control over these potential problems by managing risks during the software acquisition life-cycle.

Emerald helps manage the risks both for software acquisition organizations (buyers or acquirers) and for software development organizations (developers). At a high level, Emerald's capabilities during this process can be summarized as follows:

- Emerald derives code, use and process metrics of 3rd party suppliers and provides a risk assessment of their code and process.

- Supplier generated code can be compared with in-house code or with other supplier's code for quality, complexity, and maintainability using complexity metrics and risk models.

- Emerald gives the customer a view into the quality of the code and cost of support and maintenance.

5.3.2 Software Acquisition Overview

The Software Engineering Institute has published a Software Acquisition Capability Maturity Model (SA-CMM) [26]. This model is intended to be used to help organizations assess the maturity level of their software acquisition processes and provide guidance for making improvements. Each level consists of *key process areas* (KPAs), where each KPA defines an area to focus its improvement effort. We have derived a software acquisition life-cycle based on key process areas of the SA-CMM.

The major life-cycle phases of the acquisition are:

- Software acquisition planning
- Contract solicitation and evaluation of vendors
- Contract performance management
- Evaluation and acceptance of the resulting software products
- Transitioning to support

Software acquisition planning

Software acquisition begins with the process of defining a set of software related requirements. The software requirements should include quality and supportability requirements as well as functional and performance requirements. Acquisition planning includes schedule determination, risk identification, solicitation management and requirements definition.

Contract solicitation and evaluation of vendors

The goal of the solicitation is to select the contractor who is qualified and capable of satisfying the requirements of the contract for the least life-cycle cost. The software requirements will form the core of a solicitation package that is prepared and distributed to interested software development companies. In addition, the solicitation package should

include the estimated size and cost of the software to be developed, information on how the project will be monitored, acceptance criteria for the deliverables, information on how the offerors will be evaluated, and what documentation they must submit in order to bid on the contract. The submitted bids are evaluated according to a documented evaluation procedure and the contract is awarded.

Contract performance management

During this phase, the acquirer is responsible for tracking the contractor's performance, providing oversight, and approving new or changing requirements. As the development project proceeds the acquiring organization will need objective methods to routinely track and monitor the vendor's progress.

Evaluation and acceptance of the resulting software products

When the software is completed and tested by the vendor, the acquirer must be able to independently evaluate the software products to ensure all contractual requirements have been met. Objective, measurable methods must be available to allow the buyer to verify that the developer has met the contractual requirements. Acceptance testing is typically used to verify that functional requirements have been met. However, for all but the smallest software systems it is usually cost prohibitive to perform exhaustive testing and acceptance testing typically exercises only a small fraction of the possible program paths. According to Jones, acceptance testing only reveals about one third of the defects that remain in the product [39]. Also, if the code is poorly designed it may be difficult and

expensive to repair those defects or to modify the code later to adapt to new or changing requirements. Objective methods are needed to assess the quality characteristics of the code.

Transitioning to support

Once the software has been accepted there is a phase of transitioning the code to support by the maintenance organization. The goal of this phase is to help ensure the support organization will have the capability to understand, maintain, and support the software. The acquiring organization must be able to estimate the cost, resources, and requirements for supporting the software. The support organization must quickly learn the software architecture, design, and organization in order to support it properly.

5.3.3 Using Emerald in Software Acquisition

Emerald can be used throughout the software acquisition life-cycle to mitigate potential risks. This section describes how Emerald can be utilized in each of the acquisition phases.

Software acquisition planning

Emerald provides a measurable, quantitative approach for stating quality and support requirements. Such requirements can be used later to monitor the project as it is being developed and provide acceptance criteria for the final product. Acceptance criteria and evaluation methods should be included in the solicitation package and the resulting con-

tract. Making quality requirements explicit and measurable makes it hard for contractors to ignore and helps insure these issues will be addressed.

Contract solicitation and evaluation of vendors

Emerald can be used during the contract solicitation for evaluating the capabilities and maturity of potential bidders to be included in solicitation package. For example, bidders could be required to submit example code of recent work to assist in evaluating their capabilities.

For some acquisitions, the vendor may be providing pre-existing code to be used in constructing the new system. Emerald provides information to help assess the quality and the cost of adapting and supporting that code.

Contract performance management

Routine Emerald reviews can be used to track percent complete (as measured in code written to total estimated) and to verify that risk factors related to the code are under control. When high risk factors are revealed through these reviews, management can plan actions to mitigate those risks. Subsequent reviews can monitor the risk metrics to determine if the planned actions are working to reduce the risks.

During the course of any major software development effort it is inevitable that new or changed requirements will surface. It is often difficult to understand the consequences of making a change or introducing a new feature in a development process. Emerald can be used to help management assess the potential risks and costs involved for various imple-

mentation options for the requirement. The analysis can help determine the lowest cost implementation option, the impact it may have on schedule, and whether the proposed change is worth the anticipated benefits.

Evaluation and acceptance of the resulting software products

The code, process, and risk metrics provided by Emerald can be obtained during the evaluation and acceptance testing phase. These metrics complement the functional testing and provide an objective measure of how well the product meets quality and supportability requirements. This information can help the acquirer understand the risks and costs of accepting ownership of the resulting products.

Transitioning to support

Emerald can be useful in this phase by providing information that helps estimate and plan the resources need to support the software. Also, the metrics help the support team understand and maintain the code they are acquiring.

5.3.4 Benefits of Using Emerald in Software Acquisition

The previous section discussed the ways in which Emerald supports the software acquisition process. This section analyzes how this support leads to a number of benefits. To simplify matters, we divide these uses into two categories: *pre-award evaluation* and *project evaluation*. Figure 5.4 suggests the cause-effect chain from these uses that results in a number of benefits and Table 5.19 summarizes Emerald's benefits.

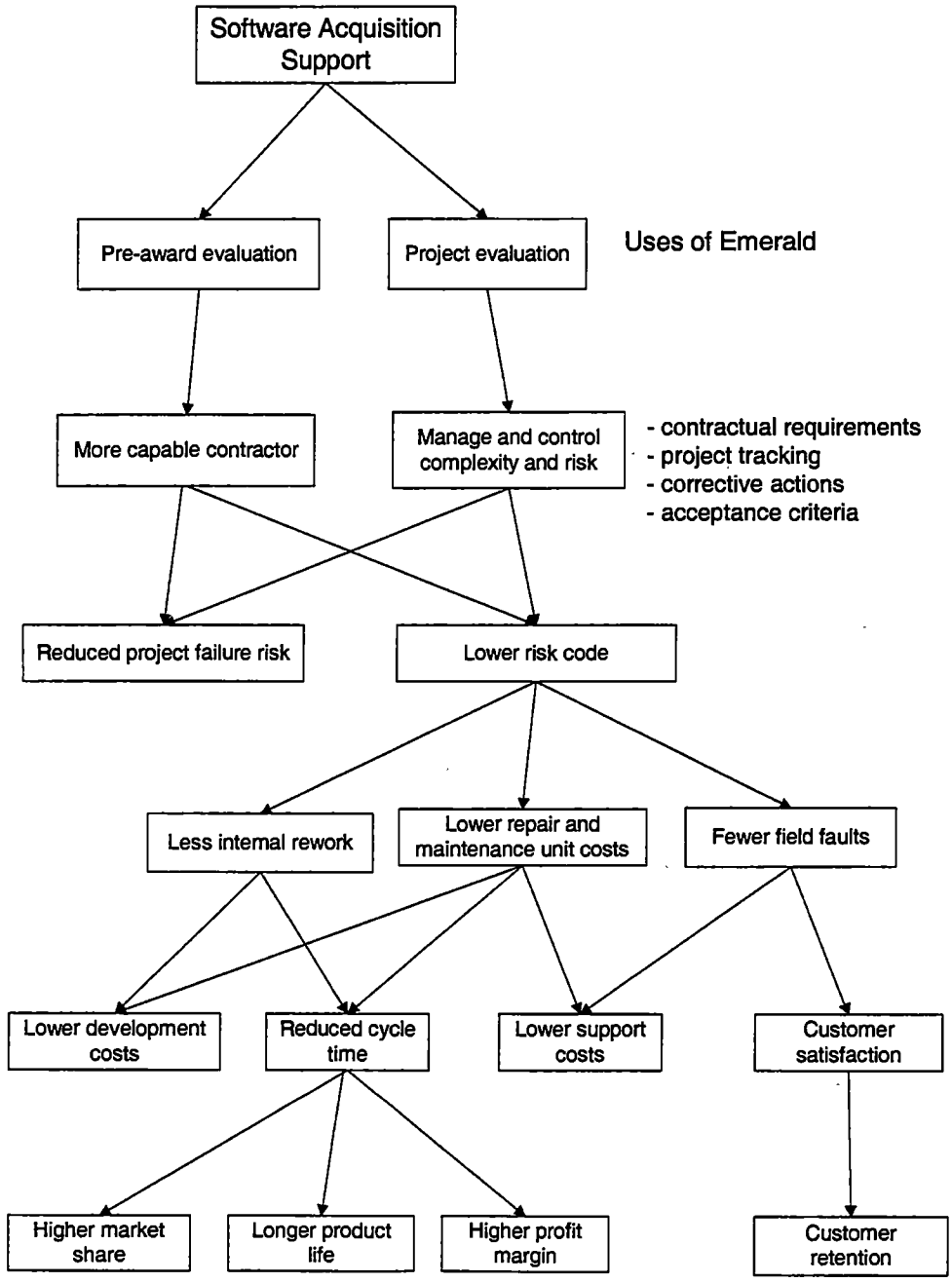


Figure 5.4: Cause-effect relationships from Emerald's software acquisition support

Table 5.19: Summary of benefits from Emerald use in software acquisition

Benefit	Rationale
Reduced acquisition risk	Emerald can help evaluate a pre-existing code base and make acquisition risks visible prior to awarding the contract. Emerald can also assist in evaluating the capabilities of a contractor.
Lower development cost	Emerald can help reduce system risk and complexity. Emerald provides management with visible metrics to help control and reduce these risks. High risk software contains a higher number of defects, requires a greater number of modifications, and those repairs and modifications require more effort.
Reduced cycle time	The use of Emerald by the buyer as a risk control mechanism can help reduce system risk. The reduction of high risk modules reduces late stage rework and thus reduces cycle time and the risk of late delivery. Reducing cycle time can result in significant financial benefits. Also, note that the use of Emerald by the contractor can help with targeted defect prevention which would reduces late stage rework thus also reducing the risk of late delivery.
Lower support cost	Reduction in field failure density Reduction in repair unit costs Reduction in maintenance unit costs
Reduction in field failure density	A reduction in system complexity reduces the faults in the code which reduces the occurrence of costly field failures.
Reduction in repair unit cost	Because Emerald can be used to help analyze failures and assess the results, and because Emerald has helped reduce system complexity, it leads to easier analysis of field failures, faster fault identification and reduced risk of introducing new faults during the repair process.
Reduction in maintenance unit cost	Software of lower risk and complexity is easier to maintain and can be more easily adapted to accommodate future business needs.
Higher customer satisfaction	Emerald results in software that contains fewer field defects and requires less time to correct problems or make changes. Customers are more satisfied when the software they use is more reliable. Also, customers appreciate fast service response when problems occur. Higher customer satisfaction encourages repeat business from existing customers and helps to attract business from new customers.

5.3.5 Quantification of Benefits

This section provides models for quantifying the costs and benefits of using Emerald for software acquisition. The benefits that can be achieved will depend on the economic characteristics of each particular acquisition and how effectively the metric information is used to improve the process.

Reduced acquisition risk

For our first model we will evaluate benefits from the point of view of estimating how Emerald can reduce the likelihood of unwanted consequences.

The unwanted outcomes of a software acquisition include:

- project is cancelled,
- project ends in litigation,
- software is delivered late,
- project exceeds development budget,
- software is of poor quality and costly to maintain or to adapt to changing requirements,
- various combinations of the above,

Our model identifies a default set of risks that can occur for the organization's software projects. For each identified risk, we must estimate the average cost (over all

projects) if the risk actually occurs. Also, we must estimate the likelihood that the risk will occur under the baseline scenario and under the process improvement scenario using Emerald. To estimate the impact of the Emerald process improvement for the entire software acquisition organization, we need the number of ongoing projects subject to these risks each year. To estimate the likelihood of risk for the Emerald software process improvement (SPI) scenario, we introduce an SPI_impact parameter that is the estimated fractional improvement in risk over the baseline case to apply to all risks. The parameters for our risk model are listed and described in Table 5.20.

Table 5.20: Software acquisition risk model parameters

Parameter	Description	Suggested data source
SPI_impact	*SPI impact.* The estimated improvement in the likelihood of the risk occurring under the software process improvement scenario.	Suggest a value in the range of 0.1 to 0.3 depending on the acquisition environment and how effectively the SPI can be applied.
num_projects	*Number of projects.* The average number of software projects subject to acquisition risks each year.	Number of projects where the SPI is to be applied.
Let Risk(0-*n*) represent a set of identified risks for the software acquisition. Let *n* be the number of Risks, Risk(*i*) represent the i^{th} Risk and Risk(i).<param> be the value of a parameter associated with the i^{th} Risk. The following parameters are required for each Risk:		
potential_cost	*Potential cost.* The total cost impact to the organization if the risk occurs.	For cost of project failure, suggest using the total cost of the project.
base_risk_likelihood	*Baseline risk likelihood.* The estimated likelihood that the risk will actually occur. For some risks, industry data is available for default values.	Historical organizational data or industry data. For example, Jones has published estimates for project cancellation.
SPI_risk_likelihood	*SPI risk likelihood.* The estimated likelihood that the risk will actually occur under the Emerald SPI scenario.	base_risk_likelihood × (1 + SPI_impact)

For each risk(i), the savings for an average project can be calculated as follows:

$$\text{project\_savings} = \text{potential\_cost} \times (\text{base\_risk\_likelihood} - \text{SPI\_risk\_likelihood}) \quad \text{(EQ 22)}$$

The savings for each risk over all projects is project_savings × num_projects. If the risks are independent, the value of using the SPI to mitigate risks over all projects can be calculated as follows:

$$\text{risk\_savings} = \sum_{i=1}^{n} \text{risk}(i).\text{project\_savings} \quad \text{(EQ 23)}$$

Table 5.21 provides an example calculation. For this example we consider two potential risks: "project cancellation" and "over 25% late delivery". Jones has published estimates of project cancellation and projects being delivered over 25% late [37]. For example, 25% of projects in the systems software subindustry category that are between 1,000 to 10,000 function points in size end up being cancelled and 21% are delivered significantly late.[1] For this example, we estimate a modest 20% improvement in the risk likelihood under the Emerald scenario (SPI_impact = 0.2) and assume the organization has an

Table 5.21: Example calculation of project risks

		Likelihood		Annual risk	
Risks	**Potential Cost**	**Baseline**	**Emerald**	**savings per project**	**Risk savings per year**
Project cancellation	$3,000,000	0.25	0.2	$150,000	$450,000
Over 25% late delivery	$500,000	0.21	0.168	$21,000	$63,000

1. Source: [37], Table 2-4, p. 60.

average of three active projects per year (num_projects = 3). The potential cost of cancellation is estimated as the average cost of a project in the organization. If the project is cancelled, no economic value can be obtained from the software and the entire cost of development has been wasted. Additional costs may be added to account for disruption of business, and lost opportunities as a result of the cancellation.

When the project exceeds its schedule by over 25 percent, the lengthened cycle time can result in penalties, lost award fees, additional resource costs, business disruption, and lost opportunities to pursue. Note that this risk reduction estimate is another way of estimating the value of cycle time reduction.

Lower development cost

The primary component of lower development cost is from a reduction in rework costs. To model this effect we obtain the average annual development cost for each software project, estimate the rework costs as a percentage of the total costs, and estimate an expected savings as a percent reduction in rework. The parameters for our model are listed in Table 5.22.

With this model the annual rework savings is computed as follows:

$$\text{annual\_rework\_savings} = \text{num\_projects} \times \text{avg\_proj\_cost\_yr} \times (\text{base\_rework\_pct} - \text{SPI\_rework\_pct}) \quad \text{(EQ 24)}$$

An example computation is given in Table 5.23 with values for the num_projects and SPI_impact parameters as before.

Table 5.22: Lower rework cost parameters

Parameter	Description	Suggested data source
SPI_impact	*Software Process Improvement (SPI) impact.* The estimated improvement in the likelihood of the risk occurring under the software process improvement scenario.	Suggest a value in the range of 0.1 to 0.3 depending on the acquisition environment and how effectively the SPI can be applied.
num_projects	*Number of projects.* The average number of software projects subject to acquisition risks each year.	Number of projects where the SPI is to be applied.
avg_proj_cost_yr	*Average project cost per year.*	Accounting data.
base_rework_pct	*Baseline rework percent.* The average percentage of cost spent on defect removal and rework.	Historical organizational data or industry data. For example, Jones has published estimates of defect removal effort for various subindustries.
SPI_rework_pct	*SPI rework percent.* The estimated percentage of cost spent on defect removal and rework under the Emerald SPI scenario.	base_rework_pct × (1 + SPI_impact)/100

Table 5.23: Example estimate of development rework savings

Average project costs per year	$3,000,000
Baseline rework percent	33%
SPI rework percent	26%
Baseline rework costs	$990,000
Emerald rework costs	$792,000
Project rework savings	$198,000
Annual rework savings	$594,000

Reduced cycle time

More capable contractor selection and more control over project risk factors increases the likelihood of reduced cycle time. Reducing cycle time can result in significant financial benefits. To quantify the benefits from reduced cycle time, either estimate the reduced risk of schedule slippage as shown in Table 5.21 or apply the model for reduced cycle time from Section 5.1.2. However, one should avoid applying both approaches as it would double count the same benefit.

Lower support cost

To calculate the value of lower support costs, we will estimate a fractional improvement in current software support costs. The parameters for this model are listed in Table 5.24. With these parameters the annual support savings are given by (EQ 25). We give an example of this model in Table 5.25.

$$\begin{aligned}\text{Annual\_support\_savings} = {} & \text{num\_projects} \times \\ & (\text{base\_maint\_cost\_yr} + \text{base\_sup\_cost\_yr} - \\ & (\text{SPI\_maint\_cost\_yr} + \text{SPI\_sup\_cost\_yr}))\end{aligned} \qquad \text{(EQ 25)}$$

Higher customer satisfaction

Higher customer satisfaction can be quantified using the approach we considered previously in Section 5.1.2.

Table 5.24: Lower support cost parameters

Parameter	Description	Suggested data source
SPI_impact	*SPI Impact.* The estimated improvement in the likelihood of the risk occurring under the software process improvement scenario.	Suggest a value in the range from 0.1 to 0.3.
num_projects	*Number of projects.* The average number of software projects subject to acquisition risks each year.	Number of projects where the SPI is to be applied.
base_maint_cost_yr	*Baseline maintenance cost per year.* Average maintenance cost per year per project for the baseline scenario.	Accounting data and project records.
base_sup_cost_yr	*Baseline support cost per year.* Average project support cost per year for the baseline scenario.	Accounting data and project records.
SPI_maint_cost_yr	*SPI maintenance cost per year.* Average maintenance cost per year per project for the software process improvement scenario.	base_main_cost_yr $\times (1 - \text{SPI\_impact})$
SPI_sup_cost_yr	*SPI support cost per year.* Average project support cost per year for the SPI scenario.	base_sup_cost_yr $\times (1 - \text{SPI\_impact})$

Table 5.25: Example estimate of support cost savings

Baseline support cost per year	$300,000
Baseline maintenance cost per year	$250,000
SPI support cost per year	$240,000
SPI maintenance cost per year	$200,000
Project support savings year	$110,000
Annual support savings year	$330,000

5.4 The Costs of Using Emerald

As with any process improvement, there are costs involved in using a tool such as Emerald. This section presents a set of effect functions for estimating the cost of using Emerald. These costs generally apply to each of the three templates that we have created for Emerald. The costs of using Emerald primarily fall under the Implementation and Support main category of the cost-benefit hierarchy in Appendix A. These costs include:

- license and maintenance fees of the Emerald product
- training
- use and operations

5.4.1 License and Maintenance Fees

The Enterprise version of Emerald is targeted for large software development projects (over 2 million lines of code). This price of this version will be approximately $150,000 and will include training, one year of support, and the consulting needed to build and verify custom risk models for the organization. This version will be fully integrated into the existing development environment and will deliver reports to the desktop via a multi-tier architecture.

A web based version of Emerald is also planned that will be targeted for organizations with a smaller code base of under 2 million lines of code or for companies with a smaller

support staff or a smaller hardware/software infrastructure. Customers will be able to download a thin Java client and use it to process their source code. It will produce metric assessment data that can be uploaded back to the Emerald web site for processing. The customer will then be able to view their risk assessment results using a Web browser. As of this writing, the pricing for the Web version has not been announced.

The software server license for Emerald Enterprise is $130,000. This price includes a custom statistical model, installation, training, and support for the first year. The price for each client license is $4,000. A minimum of five clients are sold with the server which brings the minimum configuration price to $150,000. After the first year, a support contract is available for 18% per year.

5.4.2 Training

Since the purchase of training is included in the initial purchase, the only additional training cost that needs to be considered is the time that project personnel spend in learning how to use the product. Emerald requires about four hours to train an administrator or four hours to train a user.

5.4.3 Use and Operations

This section presents equations for estimating the cost of on-going use and operations of the Emerald Enterprise system.

Operations support

Emerald requires about 2-4 staff hours per week for system administration tasks. For example, the system administrator would need to monitor the status of batch jobs and reconfigure systems as needed. The formula for estimating this cost is given by (EQ 26).

$$op\_support\_yr = 4 \times cost\_staff\_hr \times weeks\_per\_year \quad \text{(EQ 26)}$$

Use

The time required to use Emerald is will depend on specifics about how an organization plans to use the information. However, for most applications the time to use should be fairly minimal. The information is delivered to the desk top and is available as input to day-to-day decisions. The user can review the information or call-up additional detail reports to support decisions such as resource allocation or reengineering decisions. The annual cost of using Emerald (use_cost_yr) is based on estimating how many additional hours per week (use_hrs_wk) management and users will spend reviewing and analyzing the information provided by Emerald (EQ 27).

$$use\_cost\_yr = use\_hrs\_wk \times cost\_staff\_hr \times weeks\_per\_year \quad \text{(EQ 27)}$$

5.5 Validation

SPI Templates were created in the tool for each of the three uses of Emerald described in this chapter. Most effects are evaluated with hierarchical formulas. The Emerald Targeted

Defect Removal template also uses a 'Gain from early defect detection' model built into the tool to model savings that occur from improved removal efficiencies early in the life-cycle. The Support for Reengineering vs. Reuse template includes a bayesian decision model with formulas to estimate the impacts of each decision. The Support for Software Acquisition template provides models for estimating risk reduction.

The targeted defect reduction model has been used by Nortel to develop cost-benefit analysis of Emerald for 17 software organizations representing a base of over 64,000 KLOCs of code and developing over 7,000 KLOCs of new or changed code each year. Three of the organizations represent aerospace companies with the remainder from Nortel. Nortel personnel used the tool to estimate the baselines and to develop CBAs for these organizations. Several iterations of refinements were made to the templates and to the prototype tool based on comments from the users.

The Emerald Targeted Defect Removal template is listed in Section E.1 of Appendix E. An example baseline and cost-benefit analysis using this template is listed in Section F.1 of Appendix F.

5.6 Summary of the Emerald Cost-Benefit Templates

This chapter developed cost-benefit templates for three uses of Emerald. Section 5.1 described the use of Emerald for targeted defect reduction and how to quantify those effects. Section 5.2 described how Emerald can be used to support decisions to reengineer existing code components and developed a bayesian decision model to quantify the bene-

fits of that support. Section 5.3 described the software acquisition process and how Emerald can be used to manage the risks in that process. A set of benefit effects were identified and quantified for the use of Emerald. Section 5.4 summarized the costs of using Emerald that apply to any of the three uses. Section 5.5 described our experiences in testing and validating the templates.

Chapter 6

Cost-Benefit Templates for Cleanroom

This chapter develops a set of cost-benefit templates for four Cleanroom software engineering technologies. Section 6.1 provides an overview of Cleanroom and these component technologies. Section 6.2 identifies and quantifies the costs and benefits of using Cleanroom.

6.1 Cleanroom Software Engineering

Cleanroom software engineering is a collection of principles and processes aimed at the economical production of high quality software [65]. An overview of the Cleanroom process is shown in Figure 6.1. The specification team works to develop both a functional specification and a usage specification to meet customer requirements. The functional specifications are developed using a sequence enumeration process to precisely define the required behavior. This process results in specifications that are complete, consistent, and traceably correct. The usage specification identifies and classifies software users, usage scenarios, environments, and establishes a usage model. The Increment Planning process partitions the set of specified functions into a series of increments and schedules their

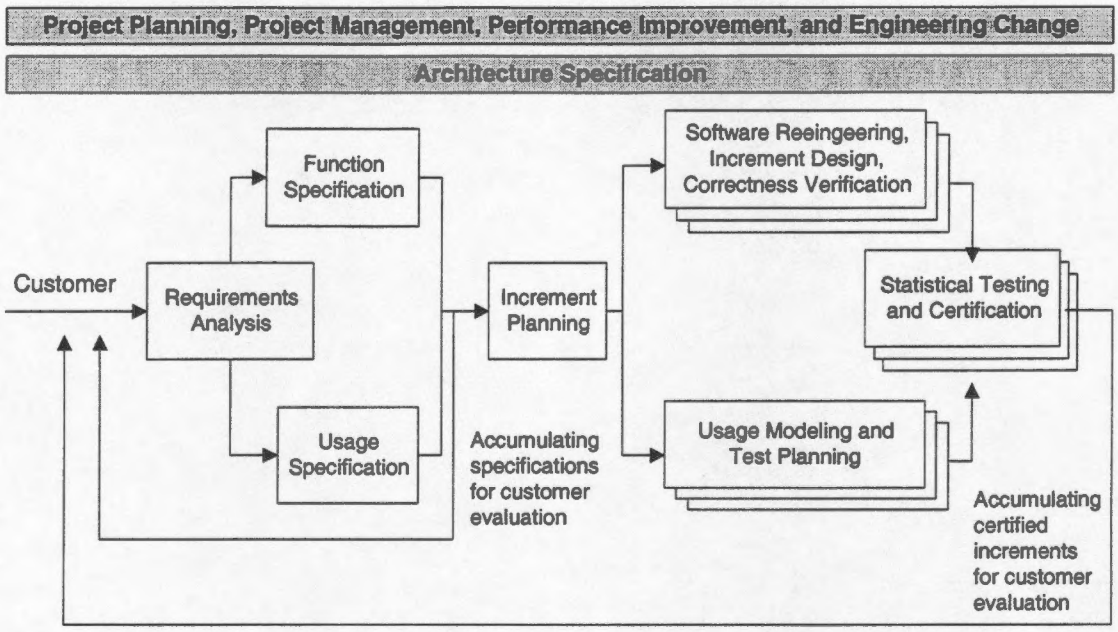


Figure 6.1: Cleanroom reference model[a]

a. Source: [65], p. 14, Figure 1.5

development and certification. The 'stacked boxes' in the subsequent process steps represent the multiple increments.

The Software Reengineering, Increment Design and Correctness Verification processes encompass developing the increment specification, reengineering existing code, design and development of new code, and correctness verification. A *black box* specification is created to define the external behavior of a system component by mapping each stimulus history to a correct response. State data is introduced to define a functionally equivalent *state box* specification. From the state box, a procedural logic is added to derive a structured *clear box* procedure. The *clear box* procedure may introduce new black boxes to represent major operations which are subsequently refined into state box and new

clear box procedures. Every black box, state box or clear box structure is subject to *correctness verification* in development team reviews. The correctness of each refinement step is verified against the previous step using reasoning based upon function theory. Correctness verification is effective at finding defects and can be used to replace unit testing and debugging.

The Usage Modeling and Test Planning process develops detailed usage models which are used to generate test cases. The Statistical Testing and Certification process executes the test cases, evaluates the results, and records failure data. The failure data is applied to a quality certification model and the resulting analysis provides feedback about the quality of the software process as well as determining if the product meets requirements.

Cleanroom consists of several component technologies that can be introduced into organizations independently. Although these technologies are designed to work together, Cleanroom processes can be independently introduced into an organization in a phased approach. This research develops economic models and cost-benefit templates for four key component Cleanroom technologies:

- Sequenced based specification
- Functional verification
- Incremental Development
- Statistical Testing

6.1.1 Sequence-Based Specification

Sequence-based specification is a systematic process for developing complete, consistent, and traceably correct software black box and state box specifications [64][65]. A black box specification defines a function that maps a response for each sequence of stimuli (sequence → response). Sequences of stimuli are enumerated in strict order beginning with sequences of length zero, length one, length two, and so on. Sequences of the same length are enumerated by a fixed lexicographic ordering. Each sequence is evaluated to determine its correct response based on user requirements. If there is no documented requirement for the response, a *derived requirement* is documented that is verified with the customer. The sequence response mapping is tagged to the appropriate requirement. If the stimulus sequence is considered impossible it is marked illegal. Each legal sequence is also checked to see if it is equivalent to some previously considered sequence. Two sequences are considered equivalent if they will yield the same response when extended for all future stimuli. Thus, it is not necessary to extend both sequences further and only the shorter of two equivalent sequences is extended. The enumeration process continues until all sequences of a given length are either illegal or equivalent to a previous sequence.

The finished black box specification is complete since the process results in all sequences being mapped to a response and it is consistent since every sequence maps to only one response. The specification is traceable to requirements that can be verified for correctness with domain experts and customers.

The *canonical sequences* are all the legal sequences in the enumeration that are not equivalent to any previous sequence. This set of sequences represents the unique condi-

tions of system usage. Canonical sequence analysis yields state data that encapsulates stimulus history. The state box specification can be represented as a function that maps the current stimulus and state to a response and state update. The completed state box specification is used for the derivation and design of the clear box.

6.1.2 Functional Verification

In Cleanroom, every software product is verified with respect to its specification. The overall black-box specification is verified against customer requirements. The state-box specification is verified against the black-box specification. The clear-box is verified against the state-box specification using function theoretic verification [48]. The Correctness Theorem defines correctness questions for every clear box control structure. A clear-box procedure is verified by verifying all constituent control structures. The rigor of the verification procedure can be adjusted depending on the risks of failure. For example to keep the verifications as cost-effective as possible verbal verifications are typically adequate. The use of standard, uniform coding practices can also help streamline the verification process.

Cleanroom work products are developed by individuals but every work product is subject to team verification reviews. However, it is usually not necessary or cost-effective for all team members to participate in the review of every work product. The allocation of resources to reviews can be determined by the risk of system failure, the risk of the work product containing defects, and how important it is that team members understand the work product and how it relates to the rest of the system.

6.1.3 Incremental Development

Incremental development organizes a large project into a series of small, manageable code development cycles. Rather than attempting to *build* a large software product through a single, long product development cycle, the developers *grow* the software incrementally over a series of smaller, cumulative cycles. The *increment planning* process partitions the required functions for a system into a series of increments for development and certification. Increments are integrated from the top down. In a new development effort the first increment typically provides a complete end-to-end framework for adding subsequent functionality. Subsequent increments will elaborate *black box* specification "stubs" in this framework.

As each increment is completed, the customer can review the system and provide feedback about the evolving product. This early feedback helps ensure the right product is being developed. Also, the performance of the team can be compared against pre-defined quality, schedule, and budget targets. Unacceptable deviations are analyzed to determine their causes. Incremental development accommodates planned adjustments to the increment plan or the development process to correct performance deviations.

6.1.4 Statistical Testing and Certification

A premise of Cleanroom testing is that it is not possible to test all possible ways in which a software system will be used, therefore, software testing is viewed as a statistical problem. As a part of this approach a set of randomly generated test cases is viewed as a ran-

dom sample obtained from an infinite population of possible software uses. The sample is evaluated in order to draw conclusions about the operational performance of the software.

In the Cleanroom approach to statistical testing, a usage model is constructed to characterize how the system will be used, and is represented as a discrete time Markov chain. The states in the chain are states of use. At any state of usage, the user has a set of possible inputs that move to the next state of use. Each of these transition arcs is labeled with an input and a transition probability that the input will occur given that the user is in the current state. The certification team constructs the initial framework structure for the usage model directly from the software specification. For example, if sequence-based specification was used, the set of canonical sequences can represent the initial state space of the usage model. Transition probabilities between states can be obtained from customer estimates or from data collected from previous versions of the software.

Once the usage model has been constructed, a number of statistics can be computed to help validate the usage model, plan the testing, evaluate the software under tests. Test cases (scripts) can be generated automatically from the usage model and can be used as input to an automatic test tool or by human testers.

6.2 Quantifying the Cost-Benefit Effects of Cleanroom

This section identifies and quantifies the cost-benefit effects from applying Cleanroom technologies. The taxonomy of Appendix A was the basis for this work. Most of the case studies published in the literature describe experiences of the general application of Clean-

room. We will first review the general costs and benefits of Cleanroom and then analyze the cost-benefit effects for the four key Cleanroom technologies.

6.2.1 Summary of Cleanroom Effects

The emphasis in Cleanroom is on building systems that are correct by design and to pursue defect prevention rather than defect removal. The construction of a black-box specification may require more time to construct than an informal specification. All Cleanroom work products are subject to team reviews which consume additional resources for both specification and development.

As a result of the more intense specification and review processes, Cleanroom requires more time and effort in specification and design phase than traditional development. In fact, design and verification activities will consume the greatest portion of the schedule for a Cleanroom project.

Extra time is required to perform the verification step, but actual time writing code decreases [54]. Because of the more intense focus on defect prevention, the Cleanroom developed code enters the testing phase with near zero defects. Cleanroom requires less time in the schedule for testing than traditional methods [32], [54]. Fewer defects remaining in the code leads to fewer cycles of rework and retesting. Also, Cleanroom statistical usage testing maximizes the increase in the operational reliability for the time spent testing [28].

The following subsections summarize Cleanroom's costs and benefits.

Costs

Costs are divided into production cost impacts and implementation costs that can be amortized over several projects.

Production Cost Impacts.

- More time is spent developing and verifying the specification and design
- More defects are found during requirements, specification and design

  Kelly [41] examined data sets for 18 products utilizing the Raytheon baseline process and 7 products utilizing Cleanroom. For the Cleanroom projects more than 50% of defects were found in requirements and design phases compared to 30% for the baseline process.
- Time and effort required for Increment Planning process
- Time and effort for Usage Specification
- Time and effort for Usage Modeling

Implementation Costs. These costs are amortized over several development projects.

- Training and coaching costs

  Sherer [71] reported that these costs added 17.3% labor to the project costs for a first time use of Cleanroom of a 90 KLOC Ada project requiring seven increments.
- Process manuals and materials
- Time to understand Cleanroom

  The "time to understand" Cleanroom is technically not a cost. However, it could be viewed as a constraint on how fast benefits will materialize. McGarry [54] stated

that the time to understand the Cleanroom methodology was approximately 26 months. However, Sherer [71] reported that benefits were realized on the very first project.

- Tool costs (license, maintenance, training)

  Although Cleanroom technologies can be introduced without them, tools make some of the Cleanroom techniques more cost-effective to apply. In particular, the technologies of statistical usage testing based on Markov chain usage models and sequence-based specification benefit from appropriate tools.

- On-going coordination and support for Cleanroom

  Most Cleanroom projects reported in the literature had staff and consultants to train participants in the technology. These responsibilities could be performed by a *software engineering process group* as defined in the context of process improvement [60]. For a small organization, team members could provide this support on a part-time basis.

Benefits

- Reduced failures in the field

  Many Cleanroom projects report dramatic reductions of errors in the field [32]. For example, Head [33] reports on a Cleanroom project at Hewlett-Packard where no errors were found during integration testing or after the product was release. He reported that Cleanroom eliminates about 99% of defects prior to release.

- Reduced error rates

  Basili observed error rates of 4.3 to 6/KSLOC versus 7/KSLOC on NASA baseline projects [6]. Hausler reported 2.3/KSLOC versus 25-35/KSLOC on traditional software (as measured from first execution of code) [32]. Linger found 3.3/ KSLOC vs. 30-50/KSLOC on traditional from first execution of code [49].

- Increased productivity

  Hausler reports productivity improvements for Cleanroom of 1.5 to 5.0 have been observed over traditional projects [32]. A specific project showed improvement of 36% more lines of code per person month. Sherer claims a productivity increase from 121 SLOC/staff month to 559 SLOC/staff month (increase of 362%) [71].

However, McGarry reported that the overall productivity was about the same as the baseline projects on a NASA case study [54].

- Reduced amount of code needed to meet functionality

  Design simplification can occur from the Cleanroom specification and verification process. For example, a Cleanroom developed prototype of an IBM COBOL Structuring Facility, estimated to require100 KLOC, was developed using only 20 KLOC [32]. However, Cleanroom is not immune to estimation errors due to initial lack of knowledge of target environments and tools or to unanticipated requirements. Hausler reported a Cleanroom project that was 49% larger than planned (from 72 KLOC to 107 KLOC) because of lack of familiarity with OS/2 Presentation Manager and unanticipated requirements [32].

- Reduced design errors and error severity

  The rigorous specification, design, and verification process reduces difficult to fix specification and design errors. Mills reported that Cleanroom errors take 20% of the time to fix [57]. Linger noted that errors found during testing or operation are simple mistakes not design mistakes [49].

- Improved defect containment

  Cleanroom is more likely to detect errors in the same phase where the errors originated. Kelly and Poore compared 18 baseline projects to 7 Cleanroom projects in an internal Raytheon study [41]. They found that for the Cleanroom projects, defects were more likely to be detected and corrected in the phase in which the defects originated. The authors estimated the improved defect containment reduced the out-of-phase rework cost by 22%.

- Time spent in testing is less

  Cleanroom case studies have shown that the testing phase is reduced for Cleanroom projects. Kelly found that Cleanroom projects required 17%-30% of software development resources compared to 32%-47% for baseline projects at Raytheon [41]. McGarry reported Cleanroom projects required 27% of the total effort compared with 30% for the baseline projects at NASA [54].

- Reduced maintenance costs

  The reduction in maintenance costs is a result of fewer defects in the field and defects that are easier to correct. Also, a Cleanroom design should prove easier to modify and adapt for new or changing requirements.

- Reduced risks

  Improved cycle-time, quality, and requirements satisfaction, reduce the risk of delay or cancellation.

- Avoiding rework

- Reduced percentage of introducing errors during rework when it does occur

- Increased job satisfaction, team spirit and team morale

  Sherer experienced this effect and attributed it to the following: a) the team now knows what to do and when to do it and how it should be done—eliminating uncertainty and anxiety; b) employees feel they finally have the tools to do high-quality work; c) Cleanroom creates reliance on team activity and fosters shared responsibility; and d) Cleanroom improves interface and communication between testing and development teams [71].

- Faster learning by new or inexperienced personnel

  Cleanroom reduces training period for new hires and the acclimation period for new project personnel as they learn and understand the system at a quicker rate from the frequent interaction of team verification reviews. Junior personnel quickly learn from their mistakes. All verification participants learn new coding techniques and design ideas from each other.

- Increased customer satisfaction

  Customer satisfaction is a secondary effect that results from better quality, more predictable schedules, and software products that meet customer requirements.

Success factors

This section describes success factors that are believed to help facilitate a successful deployment of Cleanroom. The following is a consolidated list of success factors offered by Sherer [71] and Hausler [32].

- A defined process and a technology-transfer plan

- Formal Cleanroom training
- Demonstration reviews for team education
- The use of qualified Cleanroom consultants
- Engineering handbooks
- Use of an introductory implementation
- Early and ongoing management commitment

In the sections that follow we will identify and quantify the costs and benefits of four Cleanroom component technologies.

6.2.2 Sequence-Based Specification Effects

Sequence-based specification (SBS) improves the software development process by establishing a complete, consistent, and verified specification as the foundation for remaining development and testing activities. The resulting specification helps improve coding productivity by eliminating questions or confusion about the design and what it is supposed to do. Also, the specification process results in a black box specification and a fully traceable state-box derivation that is a strong foundation for subsequent design, increment planning and test planning activities.

Costs

Production Costs.

- More time spent in specification

  The increase in specification cost (spec_incr_cost) is computed as a percentage increase (spec_incr_pct) in baseline specification cost. The baseline specification cost is estimated as a percentage (base_spec_pct) of the annual development budget (dev_budget_yr). According to Stephenson [73], specification and product design is about 15% of the development budget.

$$\text{spec\_incr\_cost} = \frac{\text{spec\_incr\_pct}}{100} \times \left(\text{dev\_budget\_yr} \times \frac{\text{base\_spec\_pct}}{100} \right) \quad \text{(EQ 28)}$$

  The value of the spec_incr_pct depends on the current baseline specification practices. A significantly higher increase will be expected for organizations that currently use informal specification approaches. For this case a 50%-100% increase is suggested. If the organization currently uses a formal specification method the increase should be minimal (e.g., 0-25%). This value also depends on the skills of the specifiers as well as the effectiveness of the tools.

Implementation Costs.

- Cost of tools to facilitate the specification process

  Tool cost consists of initial license fees, annual maintenance fees, and documentation. Usually software tools licenses are priced per client machine. For some tools there may be an additional cost for a server license. There may also be a maintenance contract at some percentage of the original license. The number of tool users (num_tool_users) can be estimated as the number of specifiers to be trained.

$$\text{tool\_cost} = \text{tool\_client\_license\_cost} \times \text{num\_tool\_users} + \text{tool\_server\_license cost} \quad \text{(EQ 29)}$$

$$\text{tool\_maint\_cost} = \text{tool\_cost} \times \text{tool\_maint\_pct} \quad \text{(EQ 30)}$$

- Personnel time in training

  This cost is estimated by the number of trainees and the training time per trainee.

$$\text{training\_time\_cost} = \text{num\_specifiers} \times \text{training\_hrs\_per\_trainee} \times \text{cost\_staff\_hr} \quad \text{(EQ 31)}$$

The default value for training hours needed per trainee is 40 hours. The number of trainees defaults to the estimated number of specifiers.

- Costs of consultants for coaching and training

  Training can be conducted in-house or off-site.Off-site training cost should include travel and lodging expenses. A default value of $1,500 is suggested for training_cost_per_trainee.

$$\text{training\_cost} = \text{num\_trainees} \times \text{training\_cost\_per\_trainee} \quad \text{(EQ 32)}$$

  Consultants can be used to help coach initial sequence-based specification efforts. Since these consultants will directly contribute to product development only a portion of their time, if any, should be counted as a cost effect.

$$\text{consulting\_cost} = \text{consulting\_hrs} \times \left(1 - \frac{\text{consult\_direct\_pct}}{100}\right) \times \text{consult\_hr\_rate} \quad \text{(EQ 33)}$$

Benefits

The sequence-based specification process results in a specification that is complete, consistent, and traceably correct [65]. These properties lead to a number of benefits as illustrated in Figure 6.2 and described below.

- Reduced specification and design rework

  Kelly reported improved defect containment in the Requirements Analysis and Preliminary Design phases [41]. His data suggests that the use of sequence-based specification plays a significant role in reducing rework in later stages. This value is calculated as a percentage reduction in annual internal failure costs.

$$\text{rework\_savings} = \text{base\_internal\_fail\_cost\_yr} \times \text{SPI\_defect\_prevent\_pct} \quad \text{(EQ 34)}$$

  where, base_internal_fail_cost_yr is the estimated baseline cost spent on internal rework each year and SPI_defect_prevent_pct is the estimated percentage of defects prevented from the use of sequence-based specification.

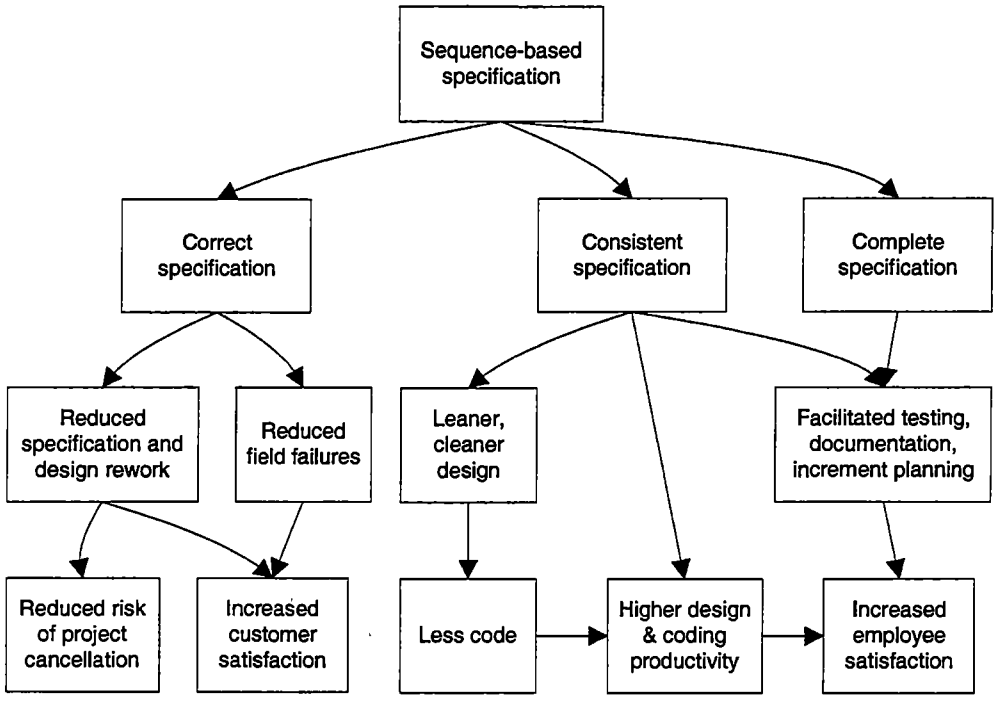


Figure 6.2: Cause-effect relationships from using sequence-based specification

- Reduced field failures

  Kelly reported that 1.4% of requirements defects escaped to and were detected in field use for Cleanroom projects, versus 5% for baseline projects [41].

  This value is calculated as a percentage reduction in annual external failure costs.

$$field\_savings = base\_external\_fail\_cost\_yr \times SPI\_defect\_prevent\_pct \quad \text{(EQ 35)}$$

- Leaner, cleaner design

  The black-box specification states precisely and completely "what" a system must do without constraining "how" it should be done. By delaying any design commitments until the specification is fully understood, the designer is more likely to choose an architecture and design representation that best fits the application.

- Less code

  A leaner, cleaner design leads to less code—less code to write, less code to document, less code to verify, less code to test and less code to maintain. Less effort for the same level of functionality contributes to higher productivity.

- Higher design and coding productivity

  Higher design productivity is a result of a complete and consistent specification. Coding productivity is higher because design, specification and requirements issues have been resolved prior to coding and because the design is more efficient.

  Higher productivity results in lower design and coding cost for a given amount of functionality. This effect can be calculated by estimating a percentage improvement in productivity (SPI_prod_imp_pct) due to the process improvement. This category should exclude productivity improvements due to rework reduction since rework reduction is counted under another category.

$$\text{higher\_prod\_savings} = (\text{design\_cost\_yr} + \text{code\_cost\_yr}) \times \left(1 - \frac{\text{rework\_budget\_pct}}{100}\right) \times \left(1 - \frac{1}{1 + \text{SPI\_prod\_imp\_pct}/100}\right) \quad \text{(EQ 36)}$$

  According to Stevenson [73], design cost is about 13% of the development budget while coding and unit testing is about 22%. To remove the rework cost we use the percentage of the budget spent on rework (rework_budget_pct).

- Facilitated testing, documentation, increment planning

  Sequence-based specification enables test planning, usage modeling, test case generation, increment planning, and documentation to proceed in parallel with design and coding activities. The specification reduces risk of miscommunication among developers, testers, and technical writers.

  This effect ultimately results in an increase in testing, documentation, and planning productivity and can be estimated in a manner similar to (EQ 36).

- Reduced risk of project cancellation

  Precise specifications are a solid foundation for the remainder of a project. By reducing rework and improving quality the project is more likely to meet schedule commitments.

  The reduced risk of project cancellation can be estimated as in Section 5.3.5.

- Increased customer satisfaction

  This is a secondary benefit that comes about from building a system that is more likely to meet all customer requirements. The value of increased customer satisfaction can be quantified by estimating an increase in business. See Table 5.11 for an example calculation.

- Increased employee satisfaction

  Employee satisfaction is a process improvement benefit that is frequently mentioned but seldom quantified. The notable exception is McGibbon who quantifies the value of improvements in employee satisfaction in terms of turnover cost savings [56]. Turnover ratios are inversely proportional to employee satisfaction. Turnover costs include recruiting costs, relocation costs, training costs, and lost performance. Not only is productivity lost while the position is being filled, but a replacement employee often requires several months of training, orientation, and experience in the new environment before achieving full productivity. During this period, the productivity is reduced of those who spend their time helping the new employee learn the job. When employee satisfaction is low, it is often the most valuable employees who leave, since they are also the most marketable to other firms. Also, low employee satisfaction may make it difficult to attract the best talent as replacements. Losing key contributors can increase development costs and risks, lengthen schedules, and lower quality.

  Our model for estimating the value of reduced employee turnover is adapted from McGibbon's analysis. The parameters for this model are listed in Table 6.1. The value of employee satisfaction, in terms of reduced turnover is given by (EQ 37).

$$\text{emp\_satisfaction\_val} = \text{dev\_staff\_size} \times (\text{staff\_turnover} - \text{SPI\_staff\_turnover}) \times \text{turnover\_cost\_emp} \quad \text{(EQ 37)}$$

Success Factors

Since the specification occurs early in the development life-cycle, SBS requires few prerequisite technologies. Appropriate tools for creating the specification can save consid-

Table 6.1: Parameters for estimating the value of reduced employee turnover costs

Parameter	Definition	Suggested data source
dev_staff_size	*Development staff size.*	Head count from personnel.
staff_turnover	*Staff turnover rate per year.*	Personnel office. Studies suggest 10-30%.
SPI_staff_turnover	*Expected staff turnover rate with SPI.*	A modest 5-10% reduction in the baseline turnover rate is suggested.
turnover_cost_emp	*Turnover costs per employee.*	This value should include costs of recruitment, relocation, training, orientation, and lost productivity.

erable time during the specification process. Detailed requirements and access to customer domain experts for clarifying requirements will be needed to develop the specification.

6.2.3 Functional Verification Effects

Functional verification has been shown to be an effective approach to eliminating defects in specifications, design, and code [32] [33] [71]. In Cleanroom, functional verification usually replaces unit testing.

Costs

Production Costs.

- Effort to perform verification process

  Verification can be applied to requirements, specifications, and design documents as well as to source code. Table 6.2 provides an example of estimating the additional cost for reviewing both documentation and code: the organization produces an estimated 3,125 function points per year representing 400,000 lines of code and 46,000 pages of documentation at a cost of $60 per staff hour.

Implementation Costs.

- Training and coaching costs

  These costs can be quantified using (EQ 31) and (EQ 32).

- Tool support for verification

  Tools are not necessary for verification but can be helpful in the management of verification status, defect tracking, and metrics collection. If tools are used, their costs can be quantified using (EQ 29).

Table 6.2: Example estimate of additional verification cost

	Documentation	Code
Output units	Pages	SLOCS
Units per FP	10	128
Units per review session	20	320
FPs per session	2.0	2.5
Baseline percent of work reviewed	40%	10%
SPI percent of work reviewed	90%	95%
SPI percent of reviews repeated	30%	20%
Administrative hours / review	1	0.5
Hours / session participant	2.5	1.5
Avg reviewers / session	3.0	3.0
Person hrs / review	8.5	5
Additional units reviewed yr	15,625	340,000
Additional FPs reviewed yr	1,563	2,656
FPs in secondary reviews	469	531
Additional review sessions / yr	1,016	1,275
Cost per review	$510	$300
Additional review hours / yr	8,633	6,375
Additional review cost / yr	$517,969	$382,500
Additional review percent of budget	5.26%	3.88%
Review cost per Unit	$33	$1.13
Review cost per FP	$332	$144

Benefits

Functional verification leads to several benefits as illustrated in Figure 6.3 and described below.

- Reduce unit testing and debugging

  Head estimated that 90% of errors on Cleanroom projects at Hewlett-Packard were eliminated by inspections with functional verification and 10% were eliminated by testing [33]. Experiments by Basili have shown that code reading by programmers other than the author is more effective and efficient at finding defects than testing [54]. Clear box verification is believed to be more effective than other kinds of team code reviews because of the use of function-theoretic reasoning.

  If an organization chooses to completely eliminate unit testing, the cost savings can be estimated by the current cost of unit testing. This cost savings is given by

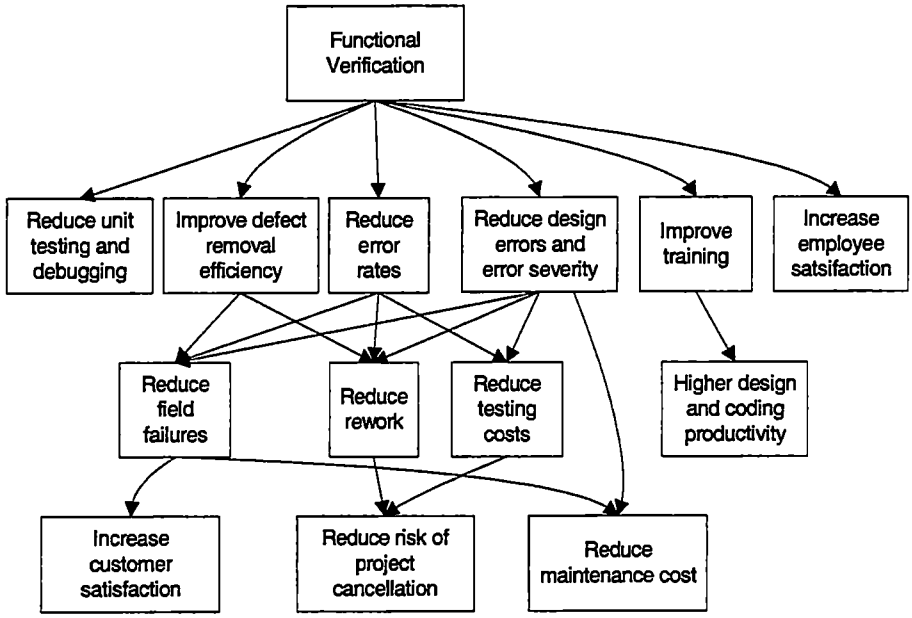


Figure 6.3: Cause-effect diagram for functional verification

$$unit\_testing\_cost = dev\_budget\_yr \times unit\_test\_pct/100 \quad \text{(EQ 38)}$$

where unit_test_pct is the percentage of the budget spent on unit testing. According to Stevenson [73], unit testing consumes about 11% of the total development effort.

Some organizations may only reduce unit testing by a percentage, thus

$$unit\_test\_savings = unit\_testing\_cost \times unit\_test\_reduce\_pct/100 \quad \text{(EQ 39)}$$

- Improve defect removal efficiency

  The most significant benefit of functional verification comes from its higher defect removal efficiency over unit testing and informal review approaches. Defects are better contained to the phase where the defect is inserted and when it is less expensive to repair.

- Reduce error rates

  See page 141.

- Reduce design errors and error severity

  See page 142.

- Improve training

  Inexperienced personnel or personnel new to the project or organization are able to quickly learn coding standards, reusable procedures, and the system architecture through the group interaction and reviews.

- Increase employee satisfaction

  Head [33] observed that the daily team verification activities "created an environment in which each person on the team took turns being in the 'hot seat.' People quickly developed an understanding that reasonable criticism was both acceptable and beneficial. The resulting frankness and openness were perceived by all to be remarkably refreshing and exhilarating." Sherer also reported improved morale and partly attributed it to the environment which "creates reliance on team activity and fosters shared responsibility" [33]. This effect can be quantified using (EQ 37).

- Reduce failures in the field

  Several Cleanroom projects report very few or no field defects [32],[33]. Head believes that Cleanroom eliminates about 99% of defects prior to release [33]. A reduction of failures in the field and reduced maintenance costs are calculated as a percentage reduction in external field failure costs.

$$\text{field\_failure\_savings} = \text{base\_external\_fail\_cost\_yr} \times \text{reduce\_ext\_fail\_pct} \quad \text{(EQ 40)}$$

- Reduced rework

  Code and design rework is eliminated as a result of improved defect removal efficiency, reduced error insertion rates, and reduced design errors. This effect is calculated as a percentage reduction in the baseline annual internal failure costs.

$$\text{rework\_savings} = \text{base\_internal\_fail\_cost\_yr} \times \text{reduce\_int\_fail\_pct} \quad \text{(EQ 41)}$$

- Reduce testing costs

  The time required for testing is less because there are fewer defects and fewer cycles of rework and retesting. We account for this effect by a percentage reduction (test_reduce_pct) in testing costs. The testing costs can be estimated as a percentage (sys_test_pct) of the development budget (dev_budget_yr). According to Stevenson [73], system testing accounts for about 15% of the development budget.

$$\text{reduced\_testing\_cost} = \text{dev\_budget\_yr} \times \frac{\text{sys\_test\_pct}}{100} \times \frac{\text{test\_reduce\_pct}}{100} \quad \text{(EQ 42)}$$

- Higher design and coding productivity

  By improving the skills of the new and inexperienced personnel, the overall development productivity is improved which helps to reduce design and coding cost. Thus, this effect is estimated by a percentage reduction (SPI_prod_imp_pct) in these costs. According to Stevenson [73], detailed design and coding (dsgn_code_pct) accounts for about 24% of the development budget.

$$\text{dev\_cost\_savings} = \text{dev\_budget\_yr} \times \frac{\text{dsgn\_code\_pct}}{100} \times \left(1 - \frac{1}{\left(1 + \frac{\text{SPI\_prod\_imp\_pct}}{100} \right)} \right) \quad \text{(EQ 43)}$$

- Increased customer satisfaction

  Verification of the specification helps insure the system meets customer requirements. Design and code verification also helps reduce field failures. The value of increased customer satisfaction can be quantified by directly estimating an increase of business that indirectly results from functional verification. See Table 5.11 for an example calculation.

- Reduced risk of project cancellation

  See the approach used for estimating the value of reduced acquisition risk in Section 5.3.5.

- Reduce maintenance costs

  Maintenance costs are reduced from fewer errors in field and from defects that are easier to correct.

Success Factors

The verification process is more effective when the product under review has been well defined. For example, it is easier to verify a design against a sequence-based specification than against an informal specification because the informal specification is likely to

be incomplete and ambiguous. The entry criteria to clear-box correctness verification is a well-defined design specification.

Programming standards to limit the allowed constructs and pre-defining correctness conditions for the standard constructs can help streamline the verification process.

6.2.4 Incremental Development Effects

The benefits of incremental development include better intellectual control, customer feedback, risk management, and facilitation of statistical process control. On the cost side, incremental development incurs an additional increment planning time to determine what functionality should go in each increment and additional regression testing as each increment after the first is certified.

To quantify incremental development effects we explore the use of a simulation model in addition to conventional equations. We use a set of equations to estimate the cost of training and the value of increased customer satisfaction, reduced cycle time, and increased employee satisfaction. To estimate the additional cost of increment planning, regression testing, and the value of reduced rework and increased productivity we offer two approaches: one using conventional equations and a second using a simulation model.

Estimating Cost-Benefit Effects Using Simulation

Our simulation model covers aspects of a single software project and contains a set of input parameters and output variables. The project size (in function points), work productivity rates (in function points per week), and staff costs (staff person cost per hour) are

examples of the input parameters for the model. The output variables allow us to measure the impact of policies on cost and schedule. The use of this model for estimating cost-benefit effects involves the following steps.

1. Build and calibrate the baseline model for a typical baseline project

   This step involves gathering data to characterize the typical project for an organization under the baseline scenario. Historic project and financial data can be used to estimate these parameters. To derive data for the typical project we use the total financial figures for the organization divided by the number of projects represented by the data.

2. Modify model input parameters for the SPI scenario

   In this case the Software Process Improvement (SPI) is incremental development. We make adjustments to the model parameters to represent the expected effects. The subsequent sections on the cost-benefit effects describe how these model parameters are adjusted for each effect under the SPI scenario.

3. Simulate and obtain results for the single project case

   We simulate models for both the baseline and SPI scenarios. The cost and schedule differences give us an estimated value for each effect.

4. Generalize the single project case to the steady state organization

   To generalize the project results to the steady state cost for an organization, we multiply the results of the simulation for a typical project by the average number of projects carried out each year by the organization.

The simulation model is described in Appendix D. The following sections describe the quantification details of each cost-benefit effect. Subsections labeled with (a) discuss how the effect is estimated using equations and subsections labeled with (b) discuss how the effect is estimated using simulation.

Costs

Production Costs.

- Time to perform increment planning step

  The purpose of Increment Planning is to allocate requirements to a series of software increments, to develop schedules and resource allocations for development, and to obtain customer agreement on the increment plan [65]. Scheduling and resource allocation are tasks that must be done with non-incremental development and thus are not considered an additional cost. The only additional cost that needs to be considered for this category is the extra time it takes to make decisions on how to allocate the functions to the increments. The extra time needed to make these decisions is relatively minor. Strategies and considerations for increment planning decisions are discussed in [74].

  (a) The effort for incremental planning represents a small increase over ordinary planning costs. We estimate the baseline effort currently being used for software development planning as a percentage (sw_plan_pct) of the development budget (dev_budget_yr). The additional cost of planning (id_plan_cost) is computed as a percentage increase (id_plan_incr_pct) in the planning budget.

$$\text{id\_plan\_cost} = \text{dev\_budget\_yr} \times \frac{\text{sw\_plan\_pct}}{100} \times \frac{\text{id\_plan\_incr\_pct}}{100} \qquad \text{(EQ 44)}$$

  The time allocated to planning and requirements gathering occupies about 6% of project resources [9]. Since this value includes requirements gathering, it should be considered an upper bound for sw_plan_pct. We suggest a default value for sw_plan_pct of 3% and a default value for id_plan_incr_pct of 10%.

  (b) To estimate this effect using simulation, we adjust the effort for incremental planning to represent the extra time for allocating functions to increments. As the number of increments increases an overhead factor increases at a more gradual slope. The overhead factor is used to lower the Plan Incr Productivity variable which has the effect of increasing increment planning cost and schedule.

- Additional regression testing

  Each increment represents the sum of all code from previous increments plus the new code. Additional testing cost is incurred since the code from previous increments is retested to some extent on each new increment.

  (a) With incremental development, the entire cumulative set of software is tested at the completion of each software increment. This regression testing represents an

additional cost for incremental development. The regression testing hours for a single project (reg_test_hrs_proj) can be estimated by

$$\text{reg\_test\_hrs\_proj} = P \times \frac{n-1}{2} \times \text{reg\_test\_hrs\_fp} \qquad \text{(EQ 45)}$$

where, P is the size of a project in function points, n is the number of increments that will be applied to a project, and reg_test_hrs_fp is the number of effort hours per function point needed for regression testing. The factor $P \times (n-1)/2$ represents the amount of code in function points subject to regression testing. Consider a sequence of n software increments where each increment adds P/n function points of new code. For each increment i, the amount of code from previous increments that is tested is $(i-1) \times P/n$ function points. The amount of code for all n increments that must be regression tested is

$$\frac{P}{n} \times \sum_{i=1}^{n} (i-1) = \frac{P}{n} \times \sum_{i=1}^{n-1} i = \frac{P}{n} \times \frac{n(n-1)}{2} = P \times \frac{n-1}{2}.$$

To estimate the additional regression testing hours for the organization, we estimate reg_test_hrs for the average software release produced during a year and multiply that result by the average number of software releases produced each year.

The parameter reg_test_hrs_fp represents the unit cost for regression testing in terms of effort hours per function point. This value should always be less than the unit cost for new function testing (new_test_hrs_fp). This unit cost will vary depending primarily on the use of automated testing. The suggested default value for reg_test_hrs_fp is 20% of new_test_hrs_fp if automated regression testing is used, or 80% of new_test_hrs_fp otherwise. The value for new_test_hrs_fp can be estimated from the hours spent on new function testing for a given amount of function points for representative projects.

The project size parameter (P) should represent the average size of a project or software release generated during a year under the baseline scenario. The parameter n can be estimated indirectly by estimating the targeted size for the average software increment avg_incr_size $< P$ under the Incremental Development scenario. Then we can estimate $n = P/\text{avg\_incr\_size}$ and rounding n to the nearest integer. The targeted avg_incr_size could represent the increment size that results in the highest productivity for the organization. Banker and Kemerer [4] offer evidence that "for most software 'production processes' there exist increasing returns to scale for smaller projects and decreasing returns for very large projects. That is,

average productivity is increasing as long as project size is smaller than the 'most productive scale size' (MPSS), and is decreasing for projects that are larger. The actual MPSS may be different for different organizations." Banker and Kemerer show how managers can estimate the most productive scale size for their organization. We hypothesize that a similar situation exists with increment size. The most productive increment size (MPIS) could be determined for an organization such that average productivity is increasing for increments smaller than MPIS and decreasing for larger increments.

Consider a project of a fixed size over a series of scenarios for decomposing the project into increments—from many small increments to few large increments. Since there is a fixed amount of overhead for each software increment the average productivity increases initially as the fixed overhead is spread over a larger increment size. Also greater productivity occurs since the increased increment size results in lower regression testing cost. Eventually, for large projects the increment size is so large that it is difficult to maintain intellectual control, and the marginal productivity of the team declines. Methods for determining the most productive increment size for an organization is a topic for further research.

(b) With the simulation model we can represent the regression testing effort through the Reg Test Flow rate shown in Figure 6.4. When an increment is delivered, the cumulative function points for that increment are added into the "To Regression Test" level. When it is time to test the next increment the "start testing" variable turns on the "reg test flow" valve to initiate the regression testing effort.

Implementation Costs.

- Training and coaching costs
- Cost of consultants to review plans

Training and coaching for incremental development is small relative to other Cleanroom technologies. To estimate these cost use (EQ 31), (EQ 32), (EQ 33).

Benefits

Incremental development provides a manageable approach to software development that leads to a number of significant benefits as shown in Figure 6.5 and described below.

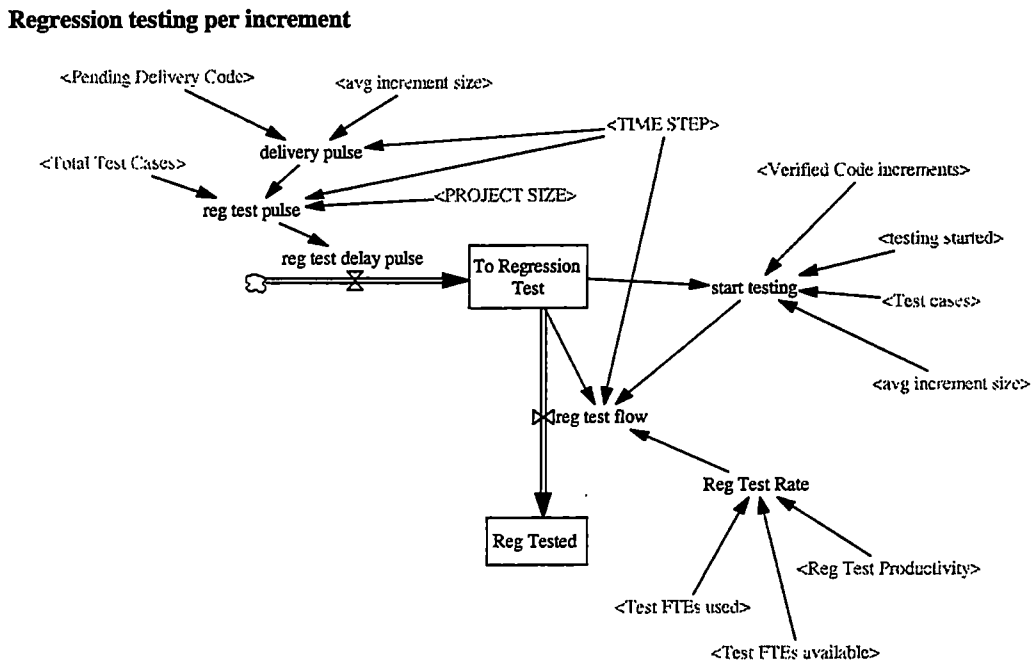


Figure 6.4: Regression testing simulation model

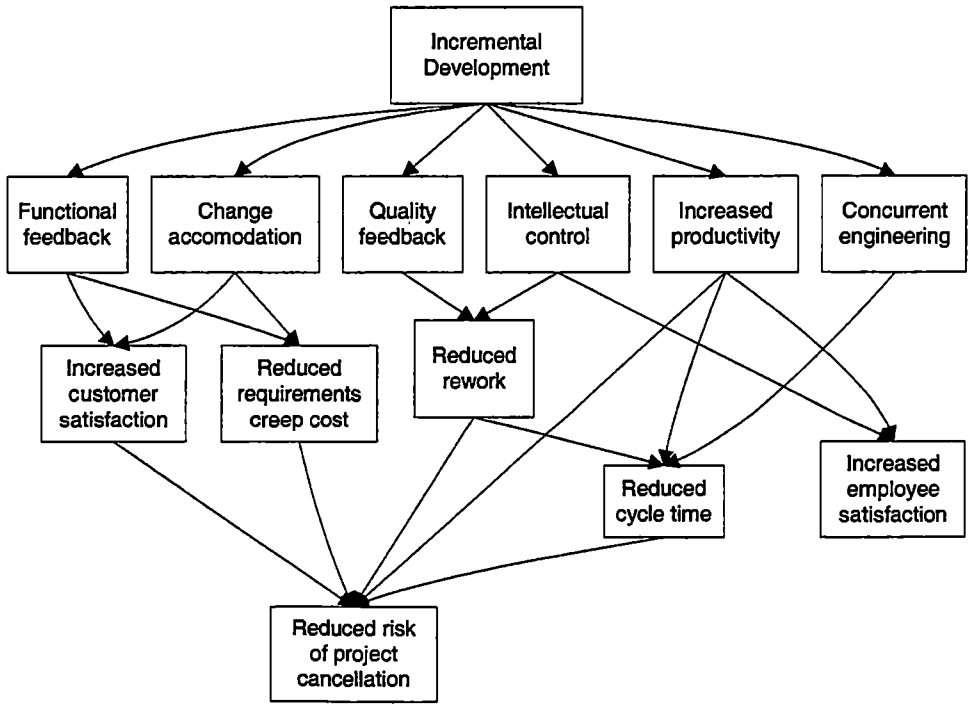


Figure 6.5: Cause-effect diagram for incremental development

- Functional feedback and change accommodation

  Incremental development allows users to examine early releases and provide feedback to ensure they are getting the system they really want (which may not necessarily be the one they asked for originally). Subsequent increments can be replanned as needed to accommodate unanticipated requirement changes. Requirement changes can be deferred to future increments reducing the harm that late requirements can cause if implemented into the middle or later part of a development effort. These effects are quantified through reduced costs of incorporating changing requirements and improvements in customer satisfaction.

- Quality feedback

  Quality measures at the end of each increment can be compared with quality standards and requirements. Smaller increments and frequent milestones result in increased visibility into the development process. Management can more frequently assess whether the process is in control and take corrective actions as needed. The value of improved quality feedback is realized through a reduction in rework.

- Intellectual control

  Incremental development supports intellectual control over the system development. The value of intellectual control is quantified through a reduction in rework and increased employee satisfaction.

- Increased productivity

  Incremental development leads to higher productivity because the deliverable goals are more manageable and attainable. Delivery date for an increment is always imminent and reduces the Parkinson's Law effect.

  (a) This effect can be estimated as a percentage improvement in coding and design productivity (SPI_prod_imp_pct) due to incremental development (EQ 36).

  (b) In the simulation model, we model this effect by increasing the productivity of development (Dev Code Productivity) and test planning (Dev Usage Model Productivity) based on the reduced size of the increment from the baseline. Default values for these productivity numbers are obtained from basic COCOMO equations [9] based on the size of the increment.

- Concurrent engineering

  Incremental development allows hardware and software to be developed in parallel. For example, both the hardware and software can be developed incrementally

in a coordinated effort. If this effect applies (e.g., for an embedded software project involving hardware development) then its value should be included in the cycle time reduction calculation (see below) by estimating additional cycle days saved due to concurrent engineering.

- Increased customer satisfaction

  Customers are able to participate in the development process and have opportunities to influence design decisions. The product from an incremental development is more likely to meet customer needs. The value of customer satisfaction can be estimated from an anticipated increase in business as shown by the example in Table 5.11.

- Reduced requirements creep cost

  The cost of incorporating new or changed requirements is reduced when incremental development is used. First, a smaller base of code and design is impacted for each requirement change requested. This requires less effort to understand what needs to be changed to accommodate the new requirements. Second, because of the smaller code base it takes less effort to modify the code and to test the changes. Third, it is less likely that changes made will introduce new defects. Finally, the requirements change is less likely to require an emergency patch. Most changes can be planned and incorporated into the next increment.

  (a) In Section 6.2.5 we provide a model for estimating the reduced cost of requirements creep that results from statistical usage modeling and testing (see (EQ 54) and Table 6.4). In that case costs are reduced by improving the requirements analysis process (through usage modeling). In the case of incremental development, we reduce the cost involved for incorporating new or changed requirements when they occur. However, (EQ 54) accommodates both scenarios since the creep reduction percentage can be used to reduce either the cost or the frequency of creep.

  (b) In our simulation, reduced cost of incorporating late requirements is represented by increasing the productivity for development rework. The "Rework Code Productivity" is increased by a percentage over the baseline value.

- Reduced rework

  The top-down approach of combining parts under intellectual control ensures all the parts fit together without rework of code from previous increments. The quality feedback provided from each increment allows the process to be improved to reduce defects and rework on subsequent increments.

  (a) Reduced rework can be quantified by estimating a percentage reduction in the baseline cost of rework. The baseline cost of rework can be estimated as a percent-

age of the development budget. Some authorities have estimated that finding and fixing errors accounts for 20-40% of the development budget [73].

$$\text{rework\_cost\_yr} = \text{dev\_budget\_yr} \times \text{rework\_budget\_pct} \quad \text{(EQ 46)}$$

$$\text{SPI\_reduce\_rework\_savings} = \text{rework\_cost\_yr} \times \text{SPI\_reduce\_rework\_pct} \quad \text{(EQ 47)}$$

(b) In the simulation model, we reduce the baseline percentages of defective code found in development and testing.

- Reduced cycle time

Higher productivity, concurrent software engineering and reduced rework help to reduce the cycle time.

(a) The value of reduced cycle time can be estimated using (EQ 58). The number of days that can be reduced from incremental development (proj_days_saved_yr) should account for time savings from increased productivity, reduced rework, and concurrent engineering.

(b) The simulation model can be used to estimate how much cycle time can be saved through incremental development, and to help determine the optimal increment size for a project. The productivity increases and reduction in rework are effects that lead to cycle time reduction. In the simulation model, cycle time is measured with the "Project Completion Time" variable as shown in Figure 6.6. The Project Completion Time is set to the Time (measured in Weeks) when the total functionality delivered to the customer is equal to the project size.

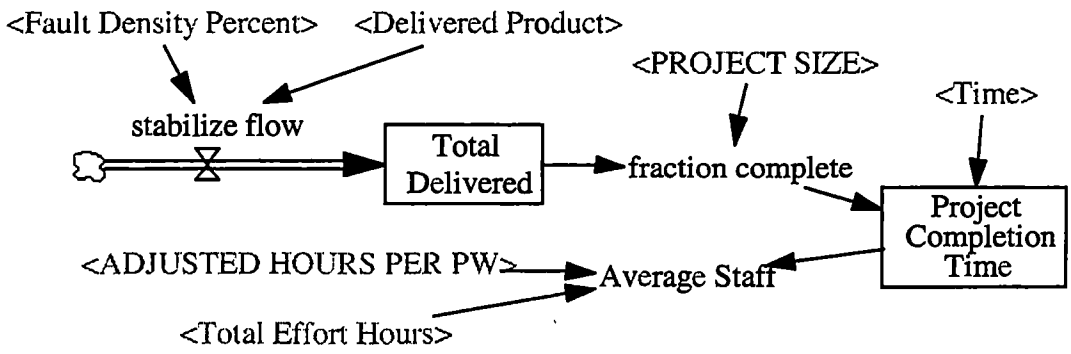


Figure 6.6: Project completion time simulation model

- Increased employee satisfaction

  Morale is higher because real progress is visible and achievable [13]. The value of avoiding employee turnover can be estimated using (EQ 37).

- Reduced risk of project cancellation

  Incremental development reduces project failure risk. Customers have better understanding of progress. Also, the product is more likely to be delivered on-time and within budget due to the reduced rework and increased productivity. The reduced risk of project cancellation can be estimated as in Section 5.3.5.

Success Factors

It is possible to begin increment planning directly from a statement of work or from a set of requirements. However, it is better to begin with a *Function Specification, Usage Specification, Software Architecture, Reuse Analysis, Risk Analysis,* and a *Schedule and Resource Plan* as defined in [65]. Guidelines for the increment planning process are in [65] and [74].

6.2.5 Statistical Testing and Certification Effects

Statistical testing involves developing a usage model in the form of a Markov chain to represent all possible ways the system can be used. The time spent creating the usage model is the most significant cost of statistical testing. This effort can be reduced by having a complete specification and appropriate tools. The benefits of statistical testing include automated test case generation, efficient testing, validation of requirements, and improved decision making.

Statistical usage testing leads to a shift in how testing resources are used. A primary effect is that less time is spent in preparing test cases and more time is spent in developing the usage specification and models. After the usage models have been created, test cases can be automatically generated from the usage models.

Our model for usage testing requires baseline information about how much the organization is currently spending on software development and testing and how much code is being generated each year.

Costs

Production Costs.

- Time to analyze, develop and maintain a usage model

  Agrawal [2] reported on the use of statistical usage testing for testing three increments of embedded, real-time software to control a tape drive. The size and effort data reported for three software increments can be described by (EQ 48). We use this equation for estimating the usage model effort required based on the code size. The cost of constructing the usage model is given by (EQ 49).

$$usage\_model\_dev\_hrs = 36 \times KLOCS\_per\_year + 15 \quad \text{(EQ 48)}$$

$$usage\_model\_cost = usage\_model\_dev\_hrs \times staff\_cost\_hr \quad \text{(EQ 49)}$$

  The process of constructing the high level usage specification can be performed prior to the functional specification. In that case we only want to count the additional time involved in documenting states of usage and estimating transition probabilities. The additional effort required to make design decisions should directly displace the functional specification effort.

Implementation Costs.

- Cost of tools for usage model development, statistical test case generation, analysis and certification, see (EQ 29) and (EQ 30)

- Time to interface usage test generator for testing environment

  To realize the efficiencies from automated testing there may be some effort involved to interface the output of the usage model to a specific testing tool or testing environment. This cost is expected to be a one time cost.

$$interface\_tool\_cost = interface\_tool\_hrs \times cost\_staff\_hr \quad \text{(EQ 50)}$$

- Personnel time in training, see (EQ 31)

- Costs of consultants for coaching and training, see (EQ 32) and (EQ 33)

Benefits

The benefits that can be derived from statistical usage testing are outlined in Figure 6.7 and described below.

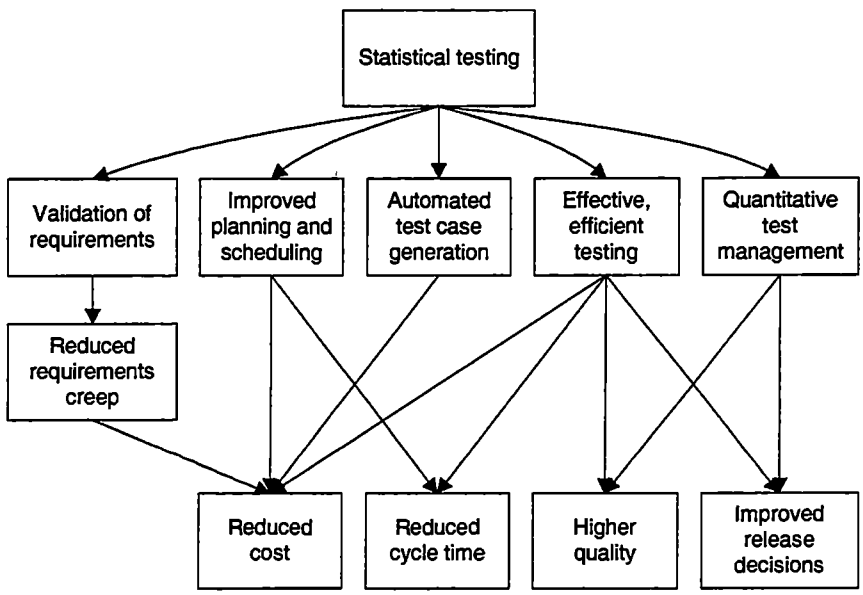


Figure 6.7: Cause-effect diagram for statistical testing

- Validation of requirements

  The process of building the usage model provides intrinsic benefits aside from providing a foundation for statistical testing. An early external view of the system is generated that can be understood and verified by developers, customers and users. Customer participation in constructing and validating the usage model helps to elicit, confirm and stabilize user requirements early in the development life-cycle. We quantify the value of this effect through the reduced risk of requirements creep (see below).

- Reduced risk of requirements creep

  Usage specification and modeling mitigate the risk of requirements creep later in the life cycle. Jones states that "defect rates associated with new features added during mid-development are about 50% greater than those of the artifacts associated with the original requirements. Defect removal efficiency levels are depressed as well, sometimes by more than 15%." He also states that "Because the costs of creeping requirements climb steeply as the development cycle proceeds, there are strong economic reasons for being very thorough early" [39].

  Our model for computing the value of the reduced risk of requirements creep is based on the idea that requirements introduced in later stages of development cost more than if they were identified during the requirements phase. Jones [39] has suggested that requirements introduced during design cost about 1.25 times as much as those introduced during the requirements phase. Table 6.3 provides example default values for the rate of growth and the relative cost of requirements.

  When usage specification is used to validate requirements, it can reduce requirements rate of growth in subsequent phases. Our model for analyzing this cost savings requires estimates for the amount of time spent in design, coding and test phases. The parameters for our model are described in Table 6.4.

Table 6.3: Requirement origins and comparative costs[a]

Development phase	Rate of growth	Relative cost
Feasibility	25%	0.75
Requirements	50%	1.00
Design	3%	1.25
Coding	1%	1.50
Testing	0.5%	2.50

a. Based on Jones [39], p. 136, Figure 1.

Table 6.4: Parameters for the value of reducing requirements creep

Parameter	Description	Example value
cost_per_FP	*Cost per function point.* The baseline life-cycle development cost per function point. This value can be estimated by dividing the development cost per year by the function points developed per year.	$1,000
FPs_per_year	*Function points per year.* The number of function points developed each year. Can be estimated by counting lines of code developed per year.	100,000
dev_budget_yr	*Development budget per year.*	
dev_phase(*i*). The following variables are needed for each major development phase after the requirements phase is complete: The development phase *i* can be either design, code, or test.		
rel_cost_fp	*Relative cost per function point.* The relative cost per function point for a requirement introduced during this phase.	1.25, 1.5, 2.5
budget_pct	*Budget percent.* The percent of the development budget spent in the design, code or test phases.	23%, 21%, 30%
creep_mth	*Creep rate per month.* The requirements creep rate per month for the development phase.	3%, 1.25%, 0.5%
creep_reduce	*Creep reduction percent.* The expected percent reduction in the creep rate due to the improvement.	10%

The cost savings from reducing requirements creep can be estimated as the difference in the cost of requirements creep with and without the process improvement.

$$\text{reduce\_rqmts\_creep\_savings} = \text{base\_rqmts\_creep\_cost} - \text{SPI\_rqmts\_creep\_cost} \quad \text{(EQ 51)}$$

The baseline cost of requirements creep is obtained from (EQ 52).

$$\text{base\_rqmts\_creep\_cost} = \text{dev\_budget\_yr} \times 12 \times \sum_i \left(\text{rel\_cost\_fp}(i) \times \frac{\text{creep\_mth}(i)}{100} \times \frac{\text{budget\_pct}(i)}{100} \right) \quad \text{(EQ 52)}$$

The cost of requirements creep with the process improvement is given by (EQ 53).

$$\text{SPI\_rqmts\_creep\_cost} = \text{dev\_budget\_yr} \times 12 \times \sum_i \left(\text{rel\_cost\_fp}(i) \times \frac{\text{creep\_mth}(i)}{100} \times \frac{\text{budget\_pct}(i)}{100} \times \left(1 - \frac{\text{creep\_reduce}(i)}{100} \right) \right) \quad \text{(EQ 53)}$$

The resulting formula for the savings simplifies to (EQ 54)

$$\text{reduce\_rqmts\_creep\_savings} = \text{dev\_budget\_yr} \times 12 \times \sum_i \left(\text{rel\_cost\_fp}(i) \times \frac{\text{creep\_mth}(i)}{100} \times \frac{\text{budget\_pct}(i)}{100} \times \frac{\text{creep\_reduce}(i)}{100} \right) \quad \text{(EQ 54)}$$

- Improved planning and scheduling

  "Standard calculations on a usage model provide data for effort, schedule, and cost projections, such as the minimum number of tests required to cover all states and transitions in the model." [65] Improved planning and scheduling increase the efficiency of the testing effort.

- Automated test case generation

  Once the usage model is built, a considerable amount of time can be saved in generating test cases. We can estimate this savings from the average time to prepare a test case by hand, the number of test cases in the baseline environment, and the cost per staff hour.

  $$\text{auto\_test\_gen\_savings} = \text{avg\_test\_case\_prep\_hrs} \times \text{num\_test\_cases} \times \text{cost\_staff\_hr} \quad \text{(EQ 55)}$$

  Alternatively, we can compute these savings by estimating the portion of the annual development budget that is currently spent on manual test case generation that would be eliminated by the usage testing.

  $$\text{auto\_test\_gen\_savings} = \text{dev\_cost\_yr} \times \frac{\text{test\_gen\_pct}}{100} \quad \text{(EQ 56)}$$

- Effective, efficient testing

  The generated test cases will test paths at the rate expected during operational use. Faults on frequently traversed paths have the highest probability of causing a failure in the field. Thus "the test budget is spent in a way that maximizes the increase in operational reliability from testing." [65]

  We estimate an improvement percentage due to statistical testing and apply it to the annual testing budget.

  $$\text{test\_eff\_savings} = \text{test\_cost\_yr} \times \frac{\text{test\_improve\_pct}}{100} \quad \text{(EQ 57)}$$

The annual system testing budget can be estimated by multiplying the annual development cost by the percent of the budget spent on testing. According to Stevenson, system testing accounts for about 15% of the development budget [73].

- Quantitative test management

  Statistical usage testing provides quantitative estimates of operational reliability as well as other quantitative results to support decisions regarding the testing process.

- Reduced testing cost

  Statistical testing reduces cost of testing by improving the test planning and scheduling process, by automating test case generation and execution, and by efficiently improving the operational reliability.

- Reduced cycle time

  Usage modeling and test generation activities can be performed in parallel with the development team without being on the critical path. When the development is complete and ready to test, testing can be performed automatically and efficiently. Also, a pre-determined quality target (MTTF) can be achieved faster. The value of cycle time reduction will depend on the industry and market conditions. We compute the value of reduced cycle time based on the number of project days saved each year times the value of each cycle day saved.

$$\text{cycle\_reduct\_value} = \text{proj\_days\_saved\_yr} \times \text{cycle\_day\_value} \quad \text{(EQ 58)}$$

- Higher quality

  Statistical testing contributes to improved field quality of the software product. The quantitative analysis and improved prediction of field quality improves process control and release decisions. Higher field quality results in lower field support costs and higher customer satisfaction.

  The primary impact of improved field quality is reduced maintenance costs and higher customer satisfaction. Maintenance cost savings can be estimated by reducing the annual maintenance cost by a percentage.

$$\text{maint\_cost\_savings} = \text{maint\_cost\_yr} \times \frac{\text{maint\_reduct\_pct}}{100} \quad \text{(EQ 59)}$$

  Customer satisfaction can be quantified as in Section 5.1.2.

- Improved release decisions

Quantitative data from the testing process can help managers make better decisions on when to stop testing and release the software. To quantify this improvement we use a bayesian decision model. Management does not know with certainty how the software will perform in the field, but must decide whether to release the software or to continue testing for a period of time. There are economic consequences of each choice depending on the true quality of the software.

A decision to release software results in several economic advantages if the quality of the software is good. The product has an advantage of getting to market earlier (possibly first) resulting in longer product life and potentially higher margins and higher market share. For government projects award fees may be won or penalties avoided for meeting release milestones. Development and testing resources can be freed to work on other projects.

However, if the software is released and the quality is poor the results can be disastrous. Development and testing resources continue to be tied up analyzing and making corrections. Serious defects that destroy user data or prevent critical features from working properly can lead to a costly product update patch, recall, or litigation. Poor quality software tarnishes the image of the product and leads to lower market share, lower margins, or both.

Development resources are used to review and rework code if significant defects are found. The delay has a cost, but it may have mitigated the higher cost of a product recall.

However, if no more serious defects are found and corrected during the extra testing, the cost of the additional testing may not have been worth the delay in the product release.

With two decision choices (release or keep testing) and two unknown states of "nature" (poor or good quality) we have four potential outcomes as shown in Table 6.5.

The values in this matrix represent the average payoff for each release decision and can be estimated by reviewing past experience for an organization. For example, all release decisions over the past year could be reviewed, categorized, and averaged according to this matrix categorization. Suppose the company correctly predicts the software field quality (and thus makes the correct decision) 85% of the time. By using Statistical Usage Testing they anticipate that they will be able to

Table 6.5: Payoff matrix example

Payoff	Good	Poor
release	$200,000	($420,000)
keep testing	($300,000)	($300,000)

Table 6.6: Baseline decision probabilities

Baseline	Good	Poor
release	43%	8%
keep testing	8%	43%

Table 6.7: Statistical usage testing decision probabilities

Statistical Usage Testing	Good	Poor
release	48%	3%
keep testing	3%	48%

correctly predict software field quality 95% of the time. Suppose further that the true state of the software is good 50% of the time when the release decisions are made. Then the decision probabilities for the baseline case are shown in Table 6.6 and the probabilities for the statistical usage testing case are shown in Table 6.7.

For the baseline case, the average decision costs $96,500. However, when statistical usage testing is used the average decision costs $65,500, representing a savings of $31,000. If the organization makes 5 decisions each year the annual savings is $155,000.

Success Factors

The process of developing the high level usage specification helps to define functional specifications and requirements. If functional specifications already exist it is easier to develop a user specification because decisions have already been made about how the application will flow. Conversely once a usage specification exists, it is easier to complete a functional specification. The usage specification process forces customers to specify how the application will flow. The additional effort for creating usage models is in documenting the usage states in the Markov chain format and estimating transition probabilities for each mode of use.

6.3 Validation

SPI templates were created in the CBA prototype tool for each of the four Cleanroom technologies described in this chapter. The evaluation functions for each of these templates were in the form of formulas. Many of the parameters in the formulas are also defined in terms of formulas. Most formulas are simple, but some use the bayesian decision model. As an example, the template for statistical testing is listed in Section E.2 of Appendix E.

An example baseline environment was created for each template to support validation. The baseline environments were created to be reasonable and self-consistent regarding code size, number of personnel, and budgets based upon available industry data. Having the SPI templates in place greatly simplified the task of producing a cost-benefit analysis for a hypothetical organization.

Once a baseline is described and an SPI has been chosen for evaluation, the user can immediately focus on determining appropriate values for the parameters. Default values and a variety of industry data is available to help the user determine values when the user lacks data. The parameters needed for SPI evaluation are divided into two categories: those that depend only on the baseline environment (and not on the SPI) and those that depend on the SPI. Once an SPI is chosen for evaluation, the user is prompted for the appropriate parameters. The baseline parameters are recorded with the baseline and the other parameters with the CBA-SPI evaluation. As the user provides a value for each parameter, the source for the data and any assumptions can be documented. Once all

parameters have been provided, a cost-benefit analysis report can be automatically generated. Several iterations of review, modifications to parameters, and regeneration are recommended. Effects can be removed, added, or modified as needed. Effect functions and parameter functions can be altered as needed.

An example cost-benefit analysis was performed for each Cleanroom technology template. Section F.2 of Appendix F presents an example CBA for statistical testing. The focus of effort for each CBA was on providing reasonable parameters for the effect evaluation functions. On reviewing the CBA reports, it was easy to spot effects that were unreasonable when compared to other effects for the CBA or for the size and budget of the baseline organization.

6.4 Summary of Cleanroom Templates

This chapter has developed cost-benefit templates for four Cleanroom technologies. Section 6.1 provided an overview of Cleanroom and the four component technologies of sequence-based specification, functional verification, incremental development, and statistical testing. Section 6.2 surveyed available literature to identify, justify, and quantify the primary cost-benefit effects for the Cleanroom technologies. Section 6.3 described our experiences in constructing, validating, and testing these templates.

Chapter 7

Conclusions

7.1 Research Contributions and Summary

This research has resulted in a practical economic framework for evaluating software process improvements. The framework applies the principles of cost-benefit analysis and leverages available economic models and data. Decision making is improved by providing an organized set of process improvement templates which includes identified cost-benefit effects, evaluation functions, and default values based on industry data and models.

Based upon this framework, we developed a prototype tool to support economic analysis of SPI initiatives. The tool contains a rich set of functionality to establish an economic baseline for the user's situation, explore alternatives, and build a business case for the best process improvement initiative. The prototype provides the ability to specify SPI templates that can be evaluated using built-in economic models, user-defined functions, process simulation models, and industrial data. The SPI templates, functions and data can be extended as needed.

To validate the framework, we constructed SPI templates for Emerald and Cleanroom technologies. The Emerald templates correspond to three ways the Emerald risk information can be used: to improve the efficiency of defect detection efforts, to support reengi-

neering decisions, and to improve the software acquisition process. The prototype has been used successfully on several large industrial software projects to estimate the value of using Emerald to improve their software process.

We also constructed SPI templates for four key Cleanroom technologies: sequence-based specification, functional verification, incremental development, and statistical testing. The cost-benefit effects for the Cleanroom technologies were identified, justified, and quantified based on the published literature.

As an integral part of this effort, a Cleanroom software process simulation model was developed and tested. This simulation model was used specifically to investigate the effects of incremental development. However, the model is general enough that it could be used to study other aspects of Cleanroom process improvements.

The value of our framework is in improving how decisions are made for implementing and sustaining process improvement efforts. The framework and prototype reduce barriers and costs for performing proper economic analysis. The resulting CBA can be used to communicate the value of SPI initiatives to project sponsors and upper management in a language they can understand. Also, the tool can be used to compile metrics from on-going SPI efforts to support continued funding of SPI initiatives that are contributing to the bottom line.

The software crisis continues in this country and around the world with unacceptable project failure rates, missed schedules, budget overruns, and low quality products. Improvements to the software process can address these problems. Yet 75% of software development organizations in the U.S. are at the lowest CMM level — characterized by

"chaotic development methods with little formality and uninformed project management." [39] A significant challenge to introducing process improvements is in winning the support of project sponsors and upper management. The impact of potential improvements is difficult for software managers to assess and even more difficult for sponsors to understand. Intense schedule pressure often leads to a focus on short-term gains (e.g., writing code with insufficient design, poor architecture, and no code review) at the expense of long-term losses (e.g., extended testing cycle, high rework, unacceptable defect levels). To a certain extent, the failure to implement and sustain improved practices is caused by uninformed (or out of control) management failing to understand their long-term costs and benefits. Our framework and prototype may help remedy this situation by making it easy to evaluate and visualize the economic impact of improved practices.

7.2 Directions for Future Activities and Research

There are many interesting and challenging directions in which this research can be extended. The implementation of the internet concept as described in Section 4.3 offers the possibility of a mutually beneficial collaboration between software engineering researchers and industrial organizations. The industrial clients would benefit from the economic analysis capabilities of the service and the research community would gain data and feedback for improving the models. An internet implementation could evolve into a comprehensive repository of SPI templates, economic models, and industrial data.

Many more SPI templates must be added to the framework to cover all common process improvements. Example SPIs to add include systematic software reuse, inspections, and various formal methods. As the repository increases, more functions, parameters, and models would need to be added. An ongoing challenge will be to limit the complexity of the models and the total number of parameters that must be provided. Keeping the framework practical and useful requires that we maximize the SPIs that can be evaluated with the minimal number of unique parameters.

The tool can be extended to account for risk and sensitivity analysis using the techniques described in Section 2.4.6. To handle sensitivity analysis, the SPI templates would need extensions for classifying parameters into dependent subsets, and to provide default high and low values for parameters.

Many additional cost models, SPI-specific economic models, process simulation models, and process simulation approaches can be incorporated into the framework that would be useful for evaluating SPI impacts. For example, COCOMO II [10] could be incorporated for estimating effects of software process maturity. Various reuse models [47] could be used for estimating the effects of systematic reuse. Other system dynamics models [51], and other types of process simulation models [66] could be used for modeling effects of inspections and other specific improvements.

An important area for further research is continuos empirical validation and improvement of the SPI templates. Measures and classification of the quality of the evidence backing the claimed benefit effects would be useful. As new evidence accumulates on an SPI, it can be systematically evaluated against earlier evidence for modifying the framework.

This research has also identified some areas where economic models are needed. As discussed in Section 6.2.4, research and methods are needed to help identify the most productive increment size. Incremental development is widely believed to be a very productive approach to development, but little guidance is provided in the literature on specifically how to organize and size increments. We explored the factors that contribute to economies of scale and diseconomies of scale related to increment size using simulation models. However, more empirical validation is needed.

The relationship among usage model size, software size, and development effort needs more empirical research. Since usage modeling is often performed early in the development life-cycle to help establish requirements, it can potentially help predict the size of an application early in the life-cycle.

In summary, everyone seems to agree that each organization needs to maintain baseline performance data, collect metrics, and manage quantitatively, yet few do so. This is a first step in providing the tools to help errant managers do what needs to be done.

REFERENCES

References

[1] Abdel-Hamid, T., and S. E. Madnick, *Software Project Dynamics: An Integrated Approach*, Prentice Hall, New Jersey, 1991.

[2] Agrawal, K., and J. A. Whittaker, "Experiences in Applying Statistical Testing to a Real-Time, Embedded Software System," *Proceedings of the Pacific Northwest Software Quality Conference*, Portland, Oregon, October, 1993, pp. 154-170.

[3] Arthur, L. J., *Software Evolution: The Software Maintenance Challenge*, New York: John Wiley & Sons, Inc.

[4] Banker, R. D., and C. F. Kemerer, "Scale Economies in New Software Development," *IEEE Transactions on Software Engineering*, Vol. 15, No. 10, October 1989, pp. 1199-1205.

[5] Basili, V., F. McGarry, G. Page, R. Pajerski, S. Waligora, M. Zelkowitz, "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.

[6] Basili, V., and S. Green, "Software Process Evolution at the SEL," IEEE Software, July 1994, pp. 58-66.

[7] Birk, A., P. Derks, R. van Solingen, J. Järvinen, "Business Impact, Benefit, and Cost of Applying GQM in Industry: An In-Depth, Long-Term Investigation at Schlumberger RPS," Fraunhofer Institute for Experimental Software Engineering, Germany, August 1998, IESE-Report 040.98/E.

[8] Boehm, B. W., "Software and its Impact: A Quantitative Assessment," *Datamation*, Volume 19, Number 5, May 1973.

[9] Boehm, B. W., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[10] Boehm, B. W. et. al., *COCOMO II Model Definition Manual*, Version 1.4, University of Southern California.

[11] Briand, Lionel, Bernd Freimut, Ferdinand Vollei, "Assessing the Cost-Effectiveness of Inspections by Combining Project Data and Expert Opinion," Fraunhofer Institute for Experimental Software Engineering, Germany, 1999, ISERN-99-14, also published as IESE-Report No. 070.99/E.

[12] Brodman, Judith G., and Donna L. Johnson, "Return on Investment from Software Process Improvement as Measured by U.S. Industry," *CrossTalk*, vol. 9, no. 4, April 1996.

[13] Brooks, Jr., Frederick P., *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, 1975.

[14] Bruckhaus, Tilman, "The Impact of Tools on Software Productivity," *IEEE Software*, Vol. 13, No. 5, September 1996, pp. 29-38.

[15] Campanella, Jack, et. al., *Guide for Reducing Quality Costs*, Milwaukee, Wisconsin, American Society for Quality Control, 1987.

[16] Canada, John R., William G. Sullivan, and John A. White, *Capital Investment Analysis for Engineering and Management*, Prentice-Hall, Inc., 1996.

[17] Coallier, François, Jean Mayrand, Bruno Lague, "Risk Management in Software Product Procurement," *Elements of Software Process Assessment and Improvement*, IEEE Computer Society, June 1999.

[18] Cruickshank, R. D., J. E. Gaffney, Jr., "An Economics Model of Software Reuse," *Analytical Methods in Software Engineering Economics*, Springer-Verlag, 1993, pp. 99-137.

[19] 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.

[20] Cusumano, M. A., and R. W. Selby, *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*, The Free Press, New York, 1995.

[21] Daskalantonakis, Michael K., "Achieving Higher SEI Levels," *IEEE Software*, July 1994, vol. 11, no. 4, pp. 17-24.

[22] Diaz, Michael, Joseph Sligo, "How Software Process Improvement helped Motorola," *IEEE Software*, September/October 1997, Vol. 14, No. 5, pp. 75-81.

[23] Dion, Raymond, "Process Improvement and the Corporate Balance Sheet," *IEEE Software*, July 1993, vol. 10, no. 4, pp. 28-35.

[24] Elmaghraby, Salah E., Elizabeth I. Baxter, Mladen A. Vouk, "An Approach to the Modeling and Analysis of Software Production Processes," *International Transactions on Operations Research*, Vol. 2, No. 1, 1995, pp. 117-135.

[25] Fenton, Norman, Phleeger, S. L., Glass R. L., "Science and Substance: A Challenge to Software Engineers," *IEEE Software*, July 1994, vol. 11, no. 4, pp. 86-95.

[26] Ferguson, J., et. al., *Software Acquisition Capability Maturity Model (SA-CMM) Version 1.01*, Technical Report CMU/SEI-96-TR-020, December 1996, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

[27] Forrester, Jay, *Industrial Dynamics*, The M.I.T. Press, New York, 1961.

[28] Fuhrer, D., and J. H. Poore, "On the Efficiency of Cleanroom Certification," Internal report, Department of Computer Science, University of Tennessee, January 1991.

[29] Glass, Robert L., "The Realities of Software Technology Payoffs," Communications of the ACM, February 1999, Vol. 42, No. 2., pp. 74-79.

[30] Haley, Thomas J., "Software Process Improvement at Raytheon," *IEEE Software*, November 1996, vol.13, no. 6, pp. 33-41.

[31] Hansen, Gregory A., "Simulating Software Development Processes," *Computer*, IEEE, January 1996, pp. 73-77.

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

[33] Head, G. E., "Six-Sigma Software Using Cleanroom Software Engineering Techniques," *Hewlett-Packard Journal*, June 1994, pp. 40-50.

[34] Herbsleb, James, et. al., "Benefits of CMM-Based Software Process Improvement: Initial Results," Technical Report CMU/SEI-94-TR-013, Software Engineering Institute.

[35] Hudepohl, John, et. al., "Emerald: Software Metrics and Models on the Desktop," *IEEE Software*, vol. 13, no. 5, September 1996, pp. 56-60.

[36] Juran, J. and F. Gryna, *Quality Control Handbook, 4th ed.*, McGraw-Hill, New York, 1988.

[37] Jones, Capers, *Applied Software Measurement*, McGraw Hill, 1996.

[38] Jones, Capers, *Patterns of Software Systems Failure and Success*, International Thomson Computer Press, 1996.

[39] Jones, Capers, *Software Quality: Analysis and Guidelines for Success*, International Thomson Computer Press, 1997.

[40] Jones, Capers, "Marry in Haste, Repent at Leisure: Successful Outsourcing Requires Careful Consideration and Planning," *Cutter IT Journal*, vol. 11, no. 7, July 1998, pp. 22-29.

[41] Kelly, D.P., and J.H. Poore, "From Good to Great: Lifecycle Improvements can Make the Difference," *Cutter IT Journal*, vol. 13, no. 2, February 2000, pp. 7-14.

[42] King, John Leslie, and Edward L. Schrems, "Cost-Benefit Analysis in Information Systems Development and Operation," *Computing Surveys,* vol. 10, no. 1, pp. 19-34, March 1978.

[43] Kramer, Bernd and Luqi, "Toward Formal Models of Software Engineering Processes," *Journal of Systems Software*, 1991, No. 15, pp. 63-74.

[44] Krasner, H., "Accumulating the Body of Evidence for the Payoff of Software Process Improvement," (1997 version), http://www.utexas.edu/coe/sqi/archive, also in "The Payoff for Software Process Improvement: What It Is and How to Get It," *Elements of Software Process Assessment and Improvement*, IEEE Computer Society, June 1999.

[45] Kyle, Brett, *Successful Industrial Experimentation*, VCH Publishers, Inc., 1995.

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

[47] Lim, W. C., "Reuse Economics: A Comparison of Seventeen Models and Directions for Future Research," *Fourth International Conference on Software Reuse: proceedings,* 1996, IEEE, pp. 41-50.

[48] Linger, R. C., H. D. Mills, B. I. Witt, *Structured Programming: Theory and Practice,* MA: Addison-Wesley, 1979.

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

[50] Linger, R.C., "Cleanroom Process Model," *IEEE Software*, March 1994, pp. 50-58.

[51] Madachy, Raymond J., "System Dynamics Modeling of an Inspection-Based Process," *Proceedings of the 18th International Conference on Software Engineering*, IEEE, 1996, pp. 376-386.

[52] Malan, Ruth; Kevin Wentzel, "Economics of Software Reuse Revisited," Hewlett-Packard Laboratories Technical Report, HPL-93-31, April, 1993.

[53] McChesney, I. R., "Toward a Classification Scheme for Software Process Modelling Approaches," *Information and Software Technology*, Vol. 37, No. 7, 1995, pp. 363-374.

[54] McGarry, Frank, Rose Pajerski, Gerald Page, Sharon Waligora, Victor Basili, Marvin Zelkowitz, "Software Process Improvement in the NASA Software Engineering Laboratory," Technical Report, CMU/SEI-94-TR-22, December 1994.

[55] McGibbon, Thomas, "A Business Case for Software Process Improvement," *DACS State-of-the-Art Report*, Air Force Research Laboratory, Rome, NY, http://www.dacs.dtic.mil, September 1996.

[56] McGibbon, Thomas, "A Business Case for Software Process Improvement Revised," *DACS State-of-the-Art Report*, Air Force Research Laboratory, Rome, NY, http://www.dacs.dtic.mil, 1999.

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

[58] Morse, Wayne J., Harold P. Roth, Kay M. Poston, *Measuring, Planning and Controlling Quality Costs*, National Association of Accountants, 1987.

[59] Oxenfeldt, Alfred R., *Decision Economics*, Crisp Publications, 1997.

[60] Paulk, M. C., C. V. Weber, B. Curtis, M. B. Chrissis, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, Reading, MA, 1995.

[61] Porter, Adam A., Harvey P. Sly, Carol A Toman, Lawrence G. Votta, "An Experiment to Assess the Cost-Benefits of Code Inspections," *IEEE Transactions on Software Engineering*, Vol. 23, No. 6, June 1997, pp. 329-346.

[62] Potok, Thomas E., *Development of a Quantitative Process Model for Object Oriented Software Development*, Ph.D. Dissertation, North Carolina State University, 1996.

[63] Poulin, J. S., *Measuring Software Reuse,* Addison-Wesley, 1997.

[64] Prowell, S. J., *Sequence-Based Software Specification*, Ph.D. dissertation, University of Tennessee, 1996.

[65] Prowell, S. J., C. J. Trammell, R. C. Linger, J. H. Poore, *Cleanroom Software Engineering: Technology and Process*, Addison-Wesley, 1999.

[66] Raffo, David M., and Marc I. Kellner, "Modeling Software Processes Quantitatively and Evaluating the Performance of Process Alternatives," *Elements of Software Process Assessment and Improvement*, IEEE Computer Society, June 1999, pp. 297-341.

[67] Reimer, Wiebke, Wilhelm Schafer, Thomas Schmal, "Towards a Dedicated Object Oriented Software Process Modelling Language," *ECOOP Workshops* 1997: 299-302.

[68] Rubin, Howard A., "Global Software Economics," *Cutter IT Journal*, vol. 12, no. 3, March 1999, pp. 6-21.

[69] Sassone, Peter G., William A. Schaffer, *Cost-Benefit Analysis: A Handbook*, Academic Press, New York, 1978.

[70] Sassone, Peter G., "A Survey of Cost-benefit Methodologies for Information Systems," *Project Appraisal*, vol. 3, no. 2, June 1988, pp. 73-84.

[71] Sherer, S. W., A. Kouchakdjian, P. G. Arnold, "Experience Using Cleanroom Software Engineering," *IEEE Software*, May 1996, pp. 69-76.

[72] Slaughter, Sandra A., D. E. Harter, M. S. Krishnan, "Evaluating the Cost of Software Quality," *Communications of the ACM*, August 1998, vol. 41, no. 8, pp. 67-73.

[73] Stevenson, C., *Software Engineering Productivity: A Practical Guide*, Chapman & Hall, 1995.

[74] Trammell, C. J., M. G. Pleszkoch, R. C. Linger, A. R. Hevner, "The Incremental Development Process in Cleanroom Software Engineering," *Decision Support Systems*, 17 (1996), pp. 55-71.

[75] Velez-Pareja, Ignacio, "Value Creation and its Measurement: A Critical Look at EVA", Social Science Research Network, Financial Accounting (WPS) Vol.3, No.17 May 24, 1999.

[76] Vienneau, R. L., "The Present Value of Software Maintenance," *Journal of Parametrics*, Vol. XV, No. 1, April 25, 1995, pp. 18-36.

[77] Weissfelner, Sam, "ISO 9001 for Software Organizations," *Elements of Software Process Assessment and Improvement*, IEEE Computer Society, June 1999, pp. 77-100.

[78] Weller, Edward F., "Lessons from Three Years of Inspection Data," *IEEE Software*, Vol. 10, No. 5, September 1993, pp. 38-45.

APPENDIX

Appendix A

Cost-Benefit Effects Hierarchy

This appendix gives the hierarchy used to classify the costs and benefit effects that result from implementing the software process improvements.

1.0 Implementation and Support

1.1 Tools and Information Systems

Costs for the acquisition and maintenance of tools and systems that may be needed to implement a process improvement.

1.1.1 Survey and assessment

1.1.2 Cost to buy or make product or tool

1.1.3 Cost of maintenance

1.1.4 Validation for use

1.2 Training

Costs required for training personnel in the new process improvement or associated tools.

1.2.1 Personnel time in training

1.2.2 Cost of training

1.2.2.1 Train the trainer

1.2.2.2 Administrative training

1.2.2.3 Training classes

1.2.2.4 Outside consultants and coaching

1.2.2.5 Computer based training

1.2.2.6 Training materials

1.3 Use and operations

1.3.1 Process start-up

1.3.2 Operations support

1.3.3 Data collection

1.3.4 Data summarization and reporting

1.4 Infrastructure

1.4.1 Support Group

1.4.2 User Group

1.4.3 Documentation

1.4.3.1 Standards

1.4.3.2 Procedures

2.0 Production Effect

We use this category for annual staff effort cost impacts to developing documentation and code as well as indirect management and support costs. This category omits impacts to defect detection and resulting rework and repair costs. However, other maintenance work (excluding rework) would be categorized under corresponding documentation or code categories.

2.1 Documentation

Includes electronic media and database forms of documentation as well as paper documentation.

2.1.1 Requirements

2.1.2 Specification

2.1.3 Architecture

2.1.4 Design

2.1.5 User Documentation

2.1.6 Training Materials

2.2 Code

This category is for activities that results in machine instructions and includes the time spent using a tool to generate a user interface or a database design as well as conventional source coding.

2.2.1 Processing

2.2.2 Database

2.2.3 User Interface

2.3 Management

2.3.1 Planning

2.3.2 Oversight & Tracking

2.3.3 Decision Support

2.3.4 Configuration Management

2.4 Operations Support

2.5 Installation & Training

3.0 Quality Effect

This category is for cost impacts of assessing quality in software products or handling failures that have occurred in software products.

3.1 Appraisal

3.1.1 Inspection and Verification

3.1.2 Testing

3.2 Internal failure

Cost of repairing defects found before product reaches customer.

3.3 External failure

Cost of handling and repairing product failures after delivery to customer.

4.0 Customer / Market Effect

4.1 Revenue Impact

4.1.1 Market life extension

4.1.2 Larger market share

4.1.3 Higher profit margin

4.2 Financial Risk Reduction

4.2.1 Reduced risk of litigation

4.2.2 Reduced risk of project cancellation

4.2.3 Reduced risk of financial penalties

5.0 Cycle Time

We intend to report the impact on cycle time as a percent calendar time savings without a direct dollar valuation. The financial impacts of cycle time reduction would appear under 4.1.

6.0 Other

This category is for identified effects that do not cleanly fit in to the other 5 categories:

6.1 Broad based effects - impacts quality, productivity, cycle time

6.1.1 Promote SEI Capability Maturity Model (CMM) progression

6.2 Personnel Resource

6.2.1 Improve employee morale

6.2.2 Faster ramp-up and training of project personnel

6.3 Miscellaneous

6.3.1 Showcasing the Emerald toolset and reselling, best in class

6.3.2 Ability to self regulate, self audit

Appendix B

Database Schema

This appendix provides an overview of the data model used by the CBA-SPI prototype software tool. The tool uses a relational database that can be conceptually divided into a number of subject areas or logical views. Each subject area contains a collection of related entities where each entity represents an object or concept about which we wish to store data.

- *Baseline Environment.* This subject area holds information to characterize specific baseline environments for an organization. The information includes an estimate of the organization's CMM level, the industry, size and annual budget, the size of the code base, how much the code base is expected to change or grow each year, and information about the quality appraisal and defect removal activities. See Figure B.1.

- *CBA Effects.* This subject area holds specific information about each cost-benefit analysis, links to the associated baseline environments, the set of SPI alternatives being considered, the estimated effects, and their estimated cash flows for the time horizon of the decision. See Figure B.2.

- *SPI Template.* These are entities that hold data for representing each potential software process improvement, its profile of cost-benefit effects, and parameter information for quantifying those effects. See Figure B.3.

- *Reference Information.* These entities provide industry data that is used to estimate the baseline scenarios and provide default parameters for quantification functions. See Figure B.4 and Figure B.5.

The following pages show the entity-relationship diagrams for these subject areas. Note that the rectangles represent entities, and the lines represent relationships between entities. The attributes for each entity are shown inside the rectangle with the primary key attributes appearing above the dividing line. Solid lines represent identifying relationships and the black dot signifies the "many" side of a one-to-many relationship. The diamond shape on the "one" side of a relationship indicates that the relationship is optional.

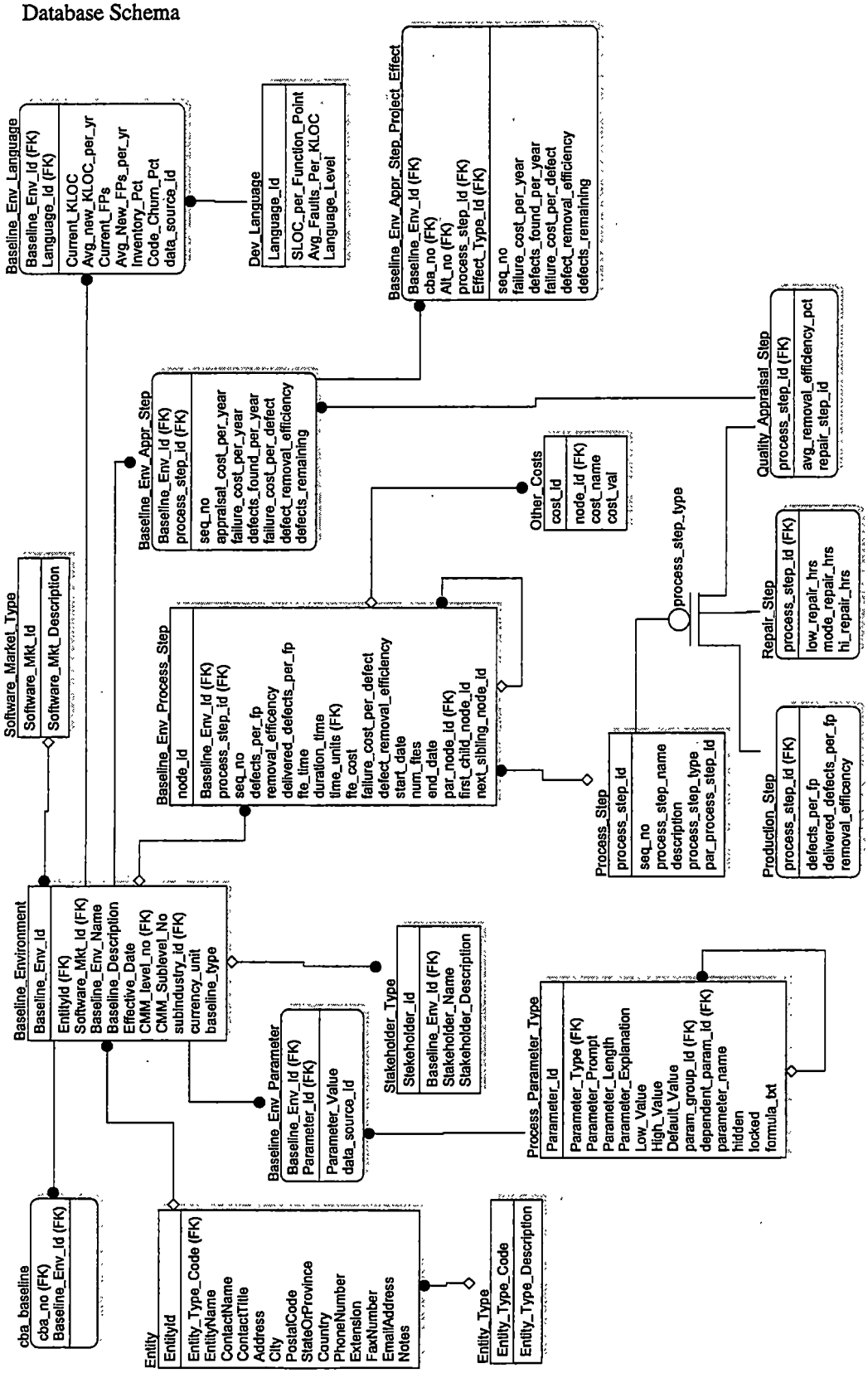


Figure B.1: Baseline environment subject area

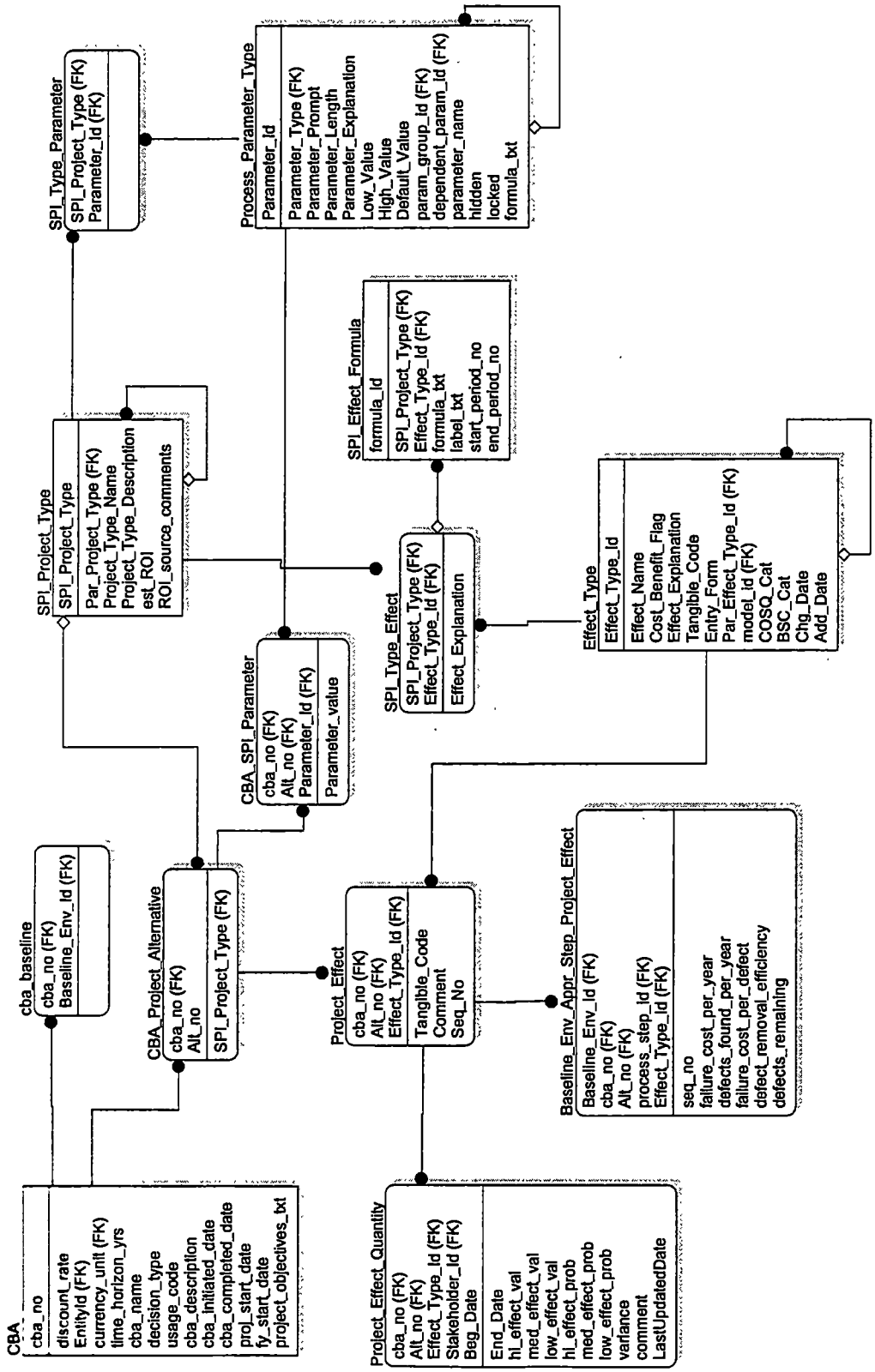


Figure B.2: CBA effects subject area

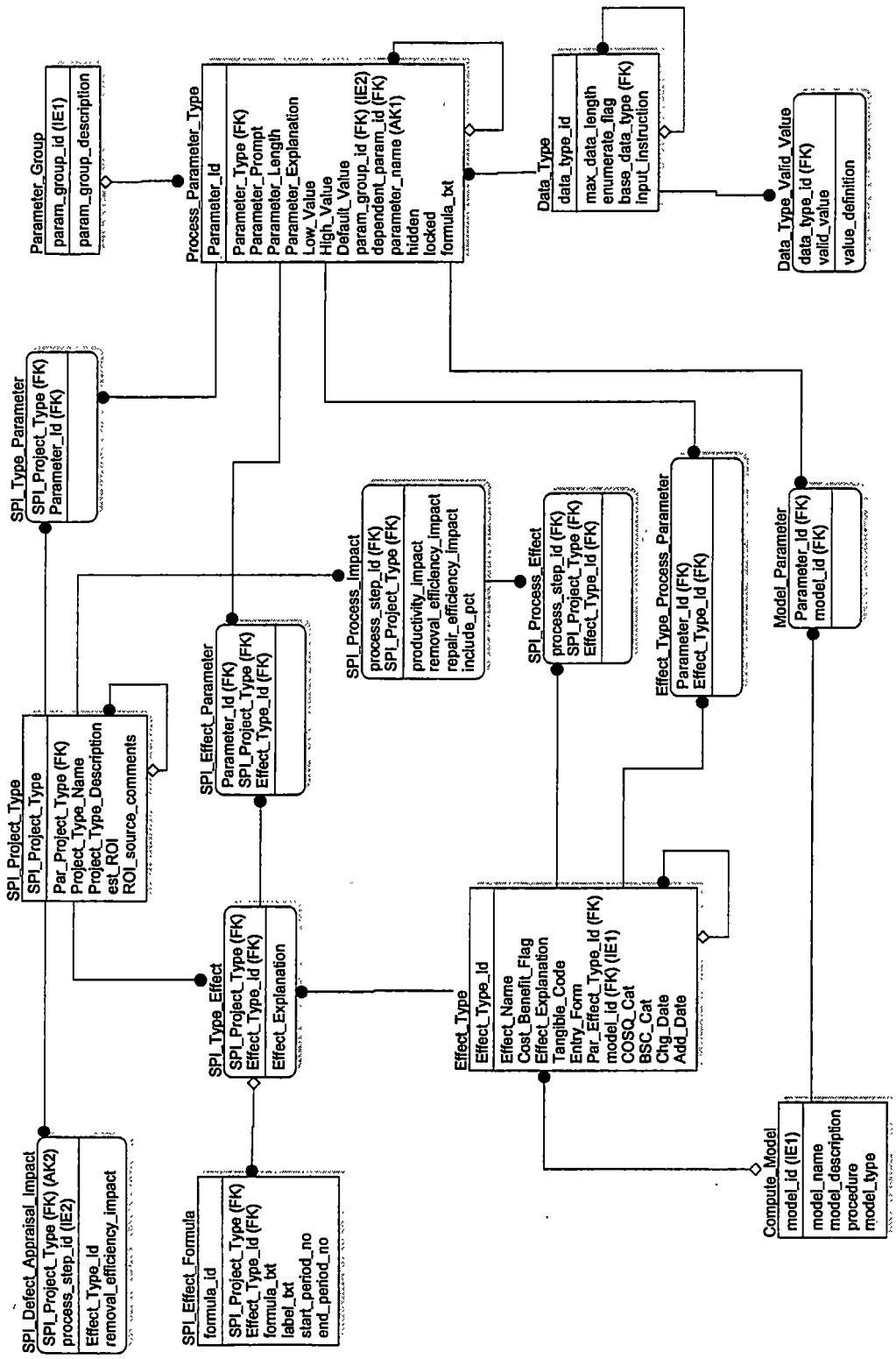


Figure B.3: SPI template subject area

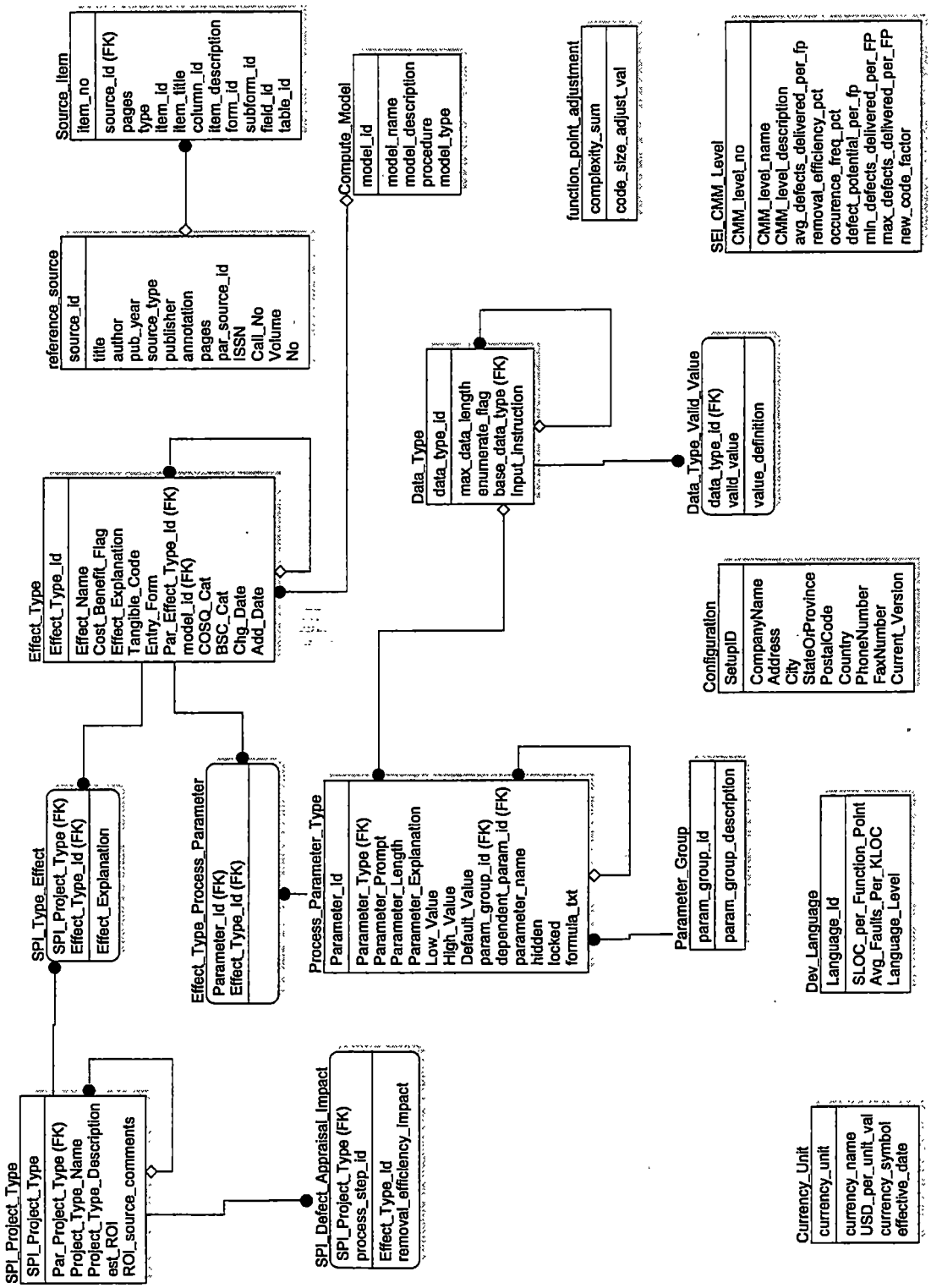


Figure B.4: Reference data subject area part 1

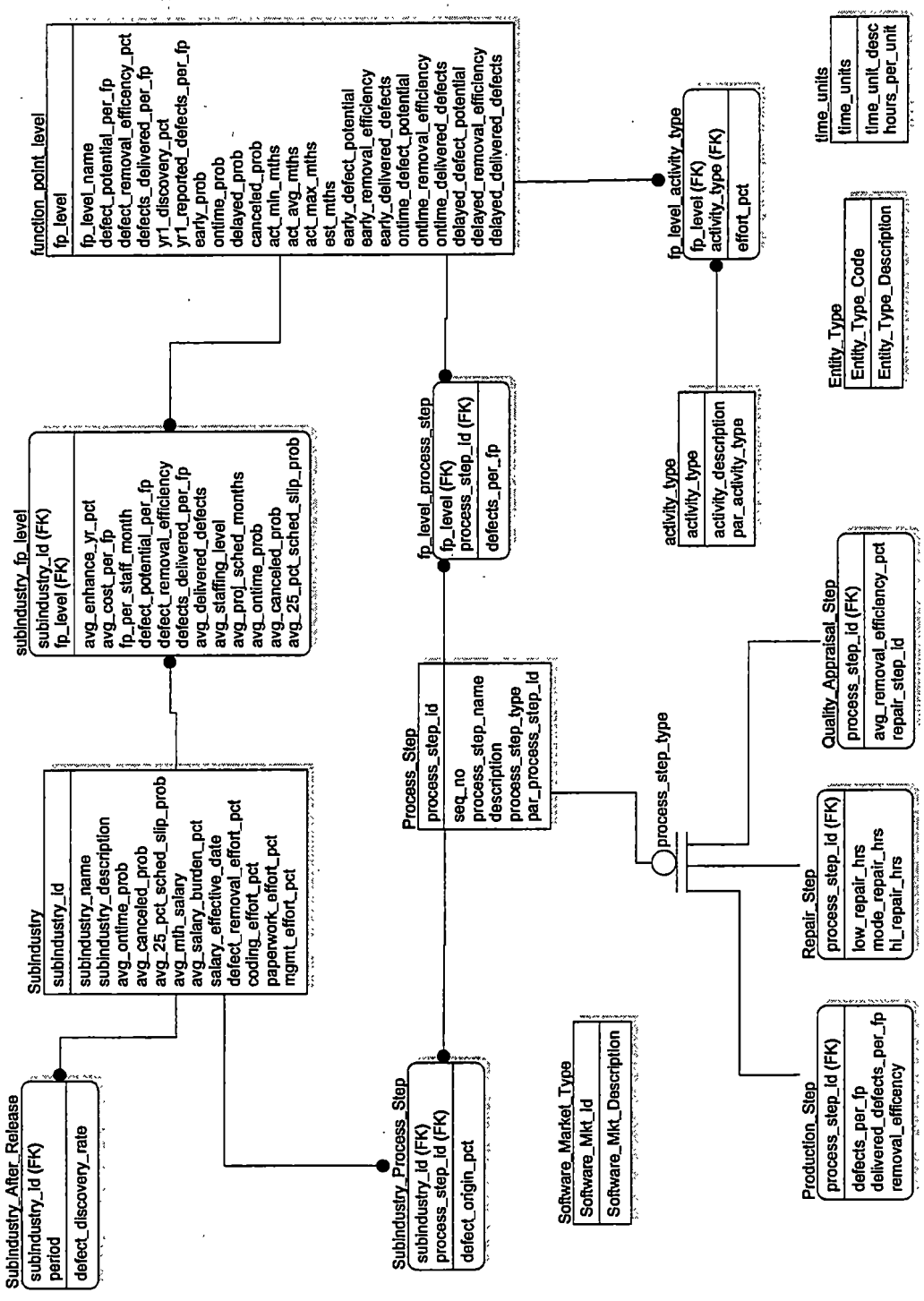


Figure B.5: Reference data subject area part 2

Appendix C

User Interface Samples

The CBA-SPI prototype includes 76 user interface forms. These user interface forms can be organized into four areas: 1) define baseline environment, 2) create a cost-benefit analysis, 3) define SPI templates, effects, parameters, and 4) provide industry data and models. This appendix exhibits one or two sample forms from each of these areas.

C.1 Define Baseline Environment

This section shows the forms used for defining a baseline environment.

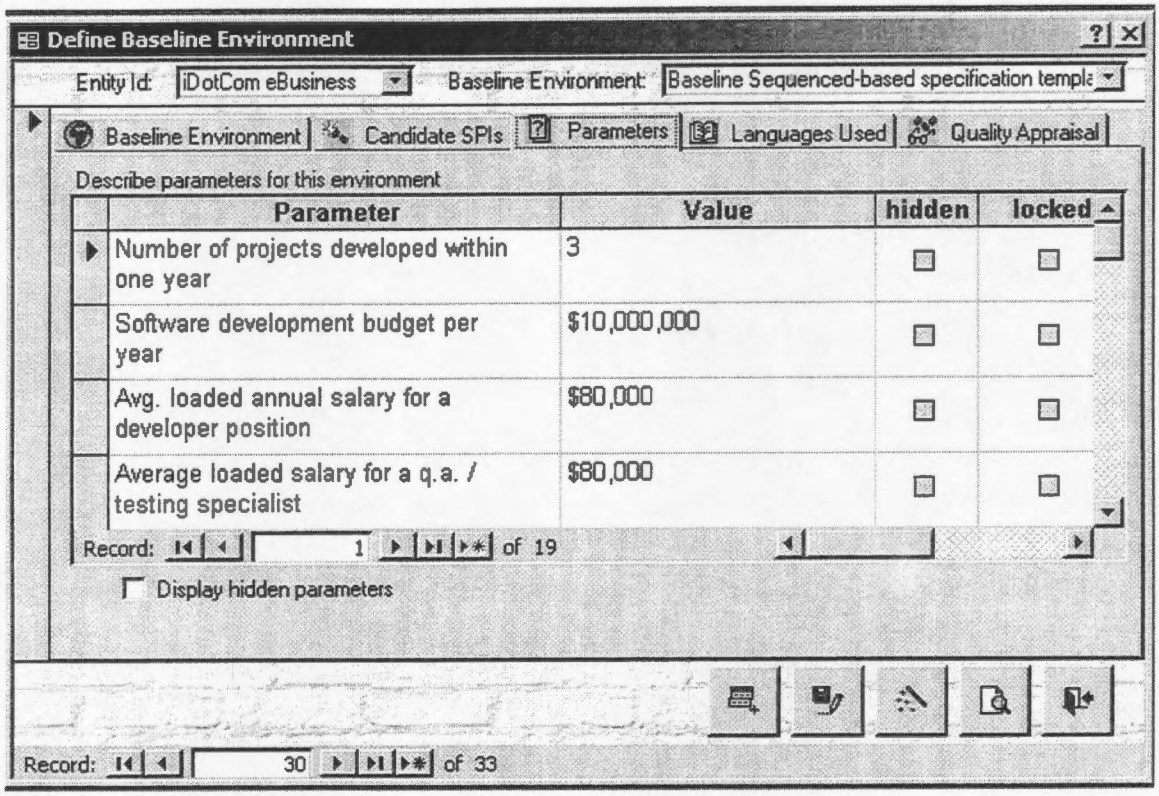


Figure C.1: Parameters for defining a baseline environment

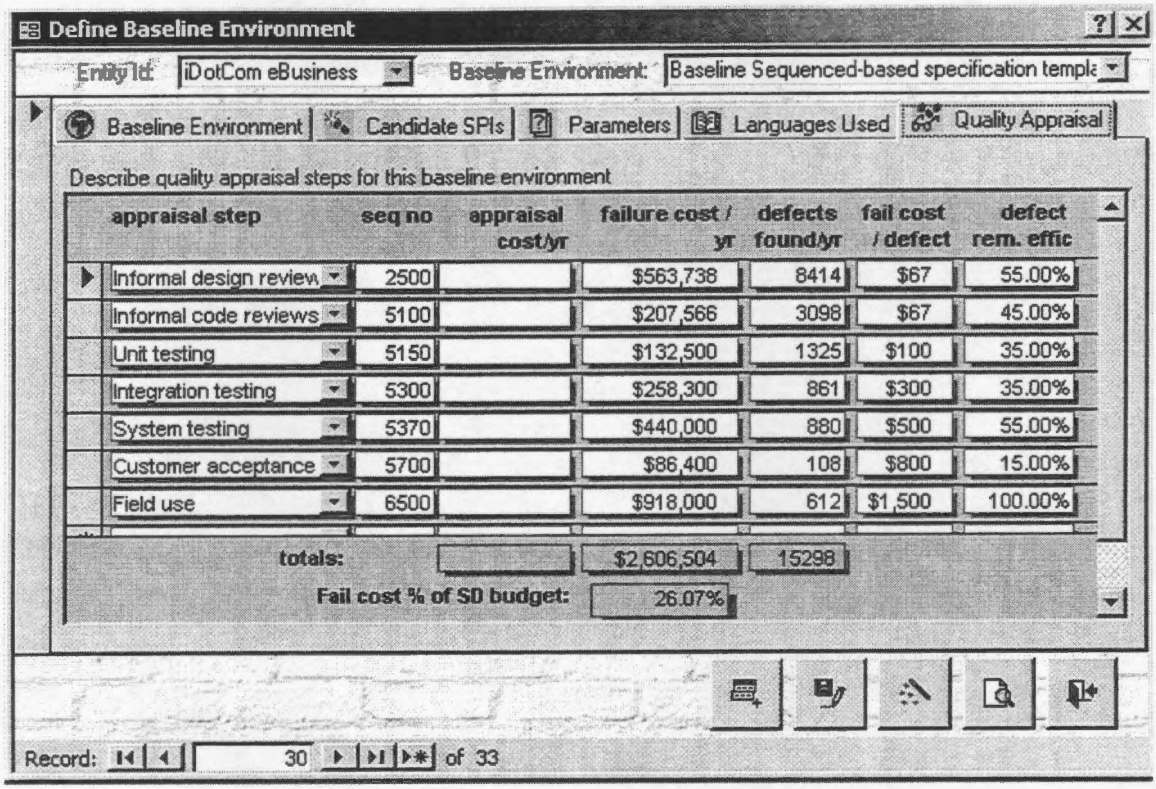


Figure C.2: Describing quality appraisal steps for a baseline environment

C.2 Create a Cost-Benefit Analysis

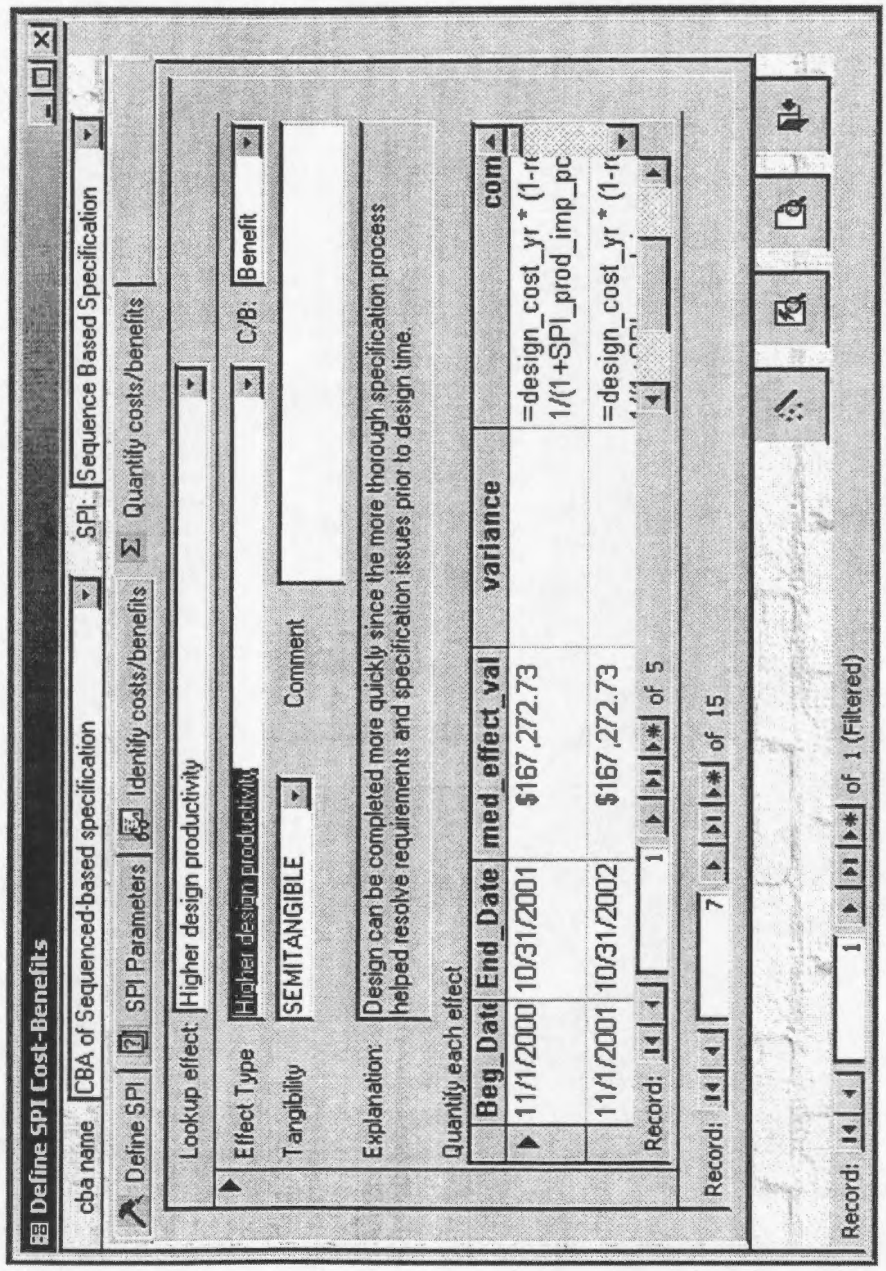


Figure C.3: View or change quantified effects

C.3 Define SPI Templates, Effects, and Parameters

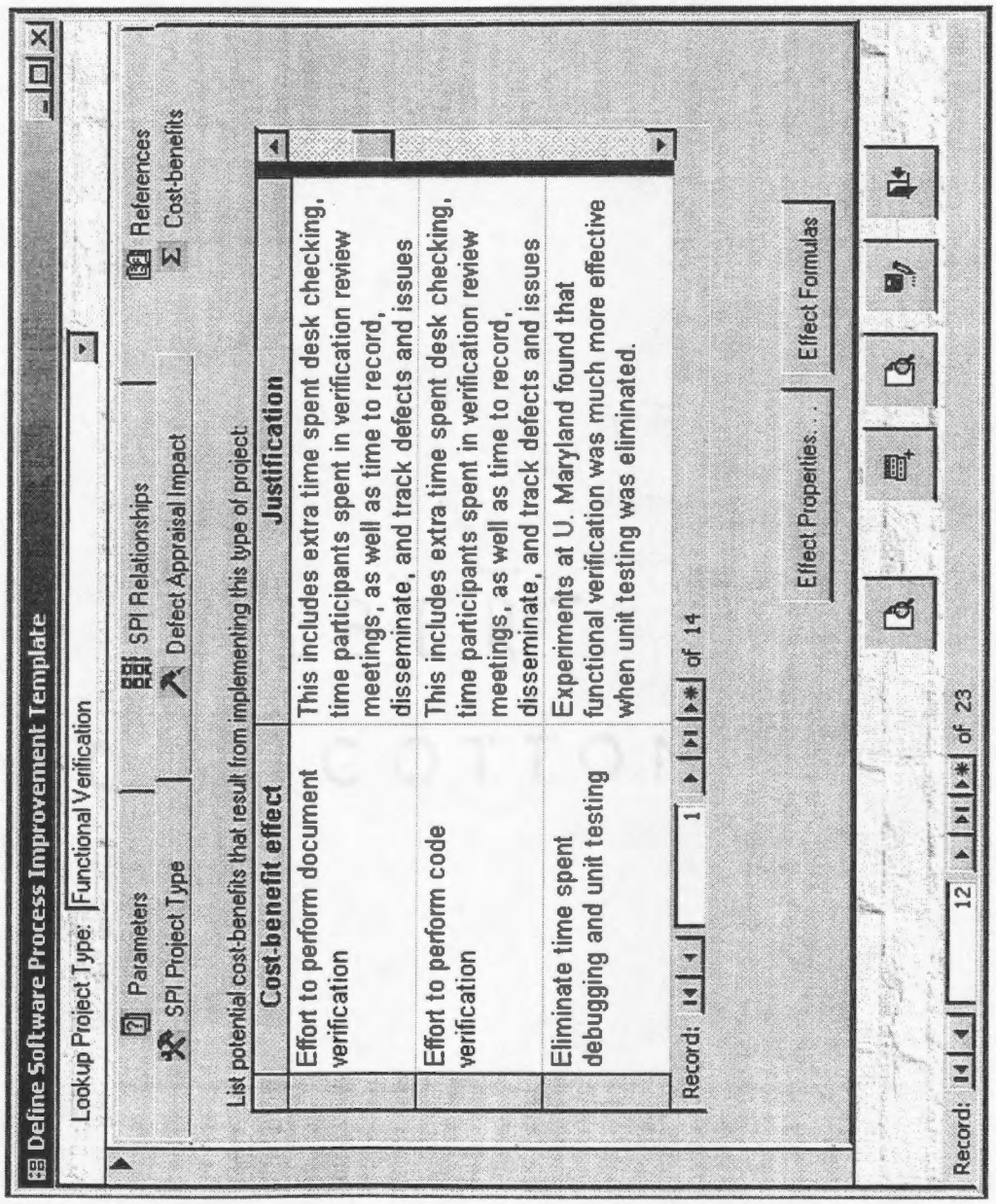


Figure C.4: Defining cost-benefit effects for the SPI template

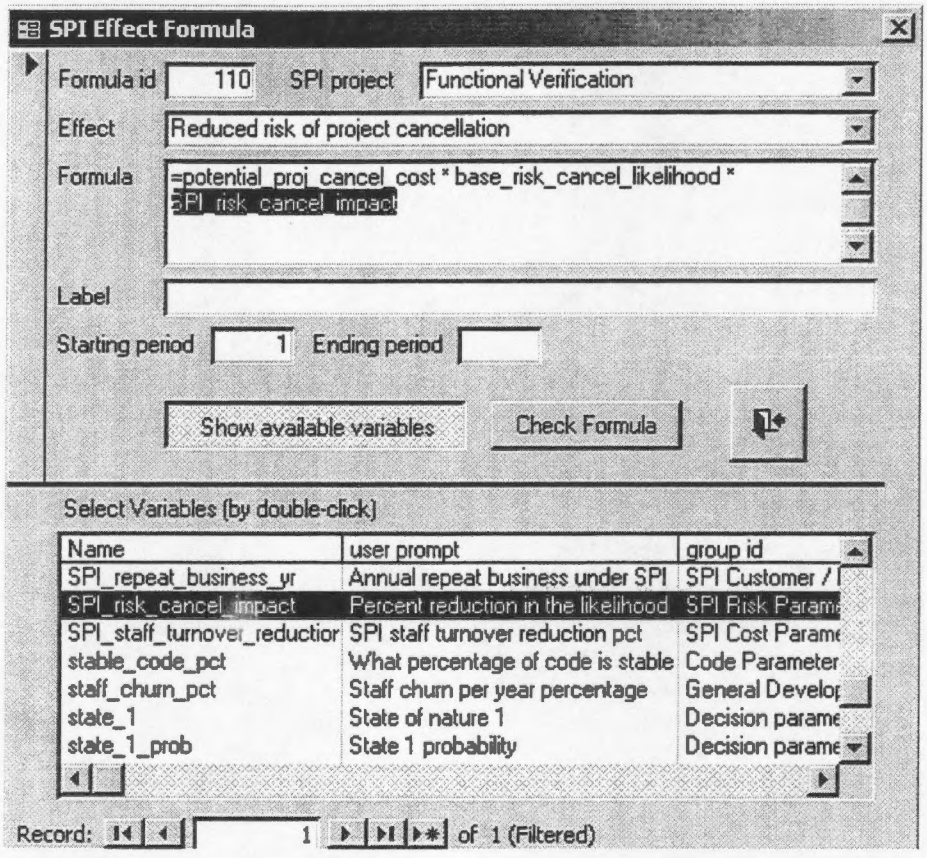
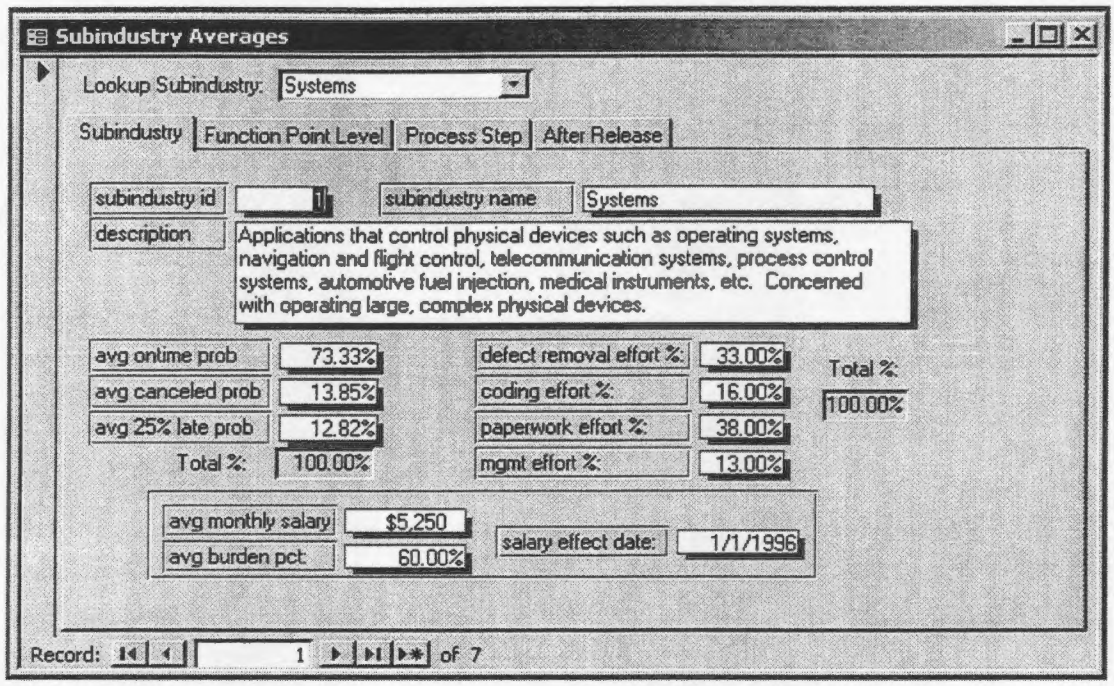


Figure C.5: Specifying a formula for evaluating a cost-benefit effect

C.4 Provide Industry Data and Models



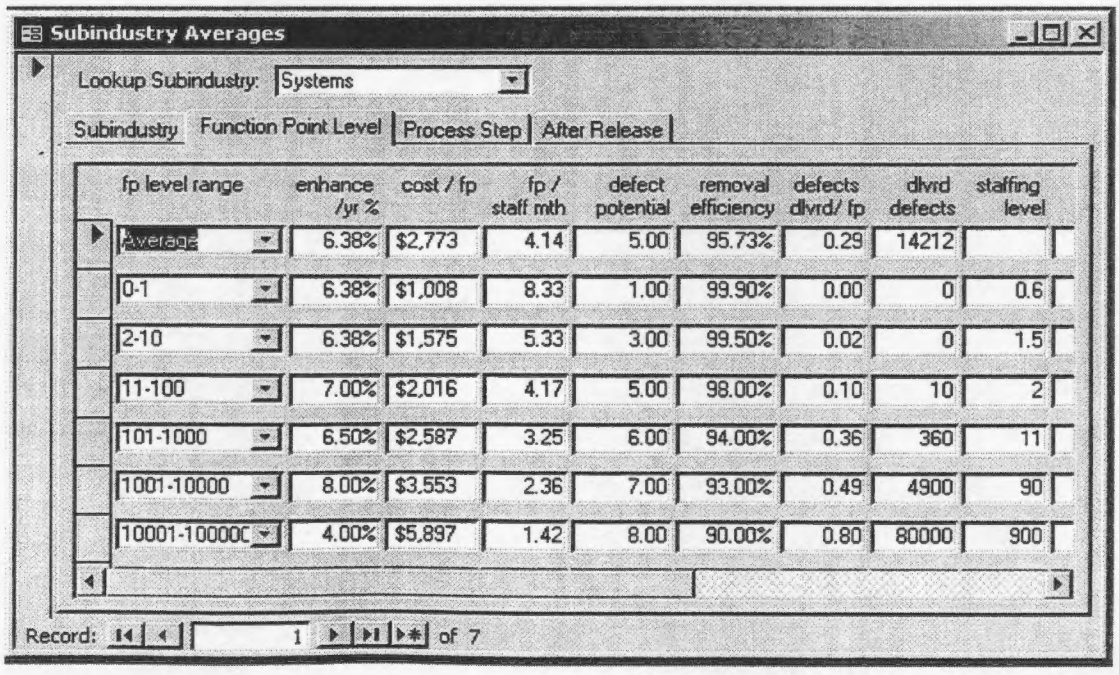


Figure C.6: Data by subindustry[a]

a. Sources for data are [37], [39]

Appendix D

Software Process Simulation Model

This appendix describes the Cleanroom software simulation model that was created as part of this research. This model was incorporated into the cost-benefit framework for analyzing the impact of incremental development as described in Section 6.2.4. Although we use the model for the specific purpose of analyzing the effects of incremental development, the model could be used for analyzing other process improvements. Also, although the model has the name Cleanroom, it can be calibrated to most software development processes for baseline calibration. We provide a brief description of system dynamics modeling notation in the next section followed by an overview of our model.

D.1 System Dynamics Modeling Concepts

There are three types of equations in a system dynamics model that are represented by three types of graphical elements: levels, flows and auxiliaries. Levels (also called stocks, states, or accumulations) are represented by rectangles. Flows (also called rates) are represented by the pipes and valves and indicate a flow between two levels. The auxiliaries are either constants, equations, or data that are used to calculate intermediate results for use in

computing rates. Auxiliary constants do not change with time and are shown in uppercase letters. Auxiliary variables represent equations that can change over time and are shown in lowercase letters. The connecting single line arrows represent dependencies between equations.

The mathematical relationships between these elements are represented by the following equations:

$$\text{levels}_T = \int_0^T \text{flows}_t \, dt$$

$$\text{flows}_t = g(\text{levels}_t, \text{aux}_t, \text{const})$$

$$\text{aux}_t = f(\text{levels}_t, \text{aux}_t, \text{const})$$

D.2 Cleanroom Process Model

A stock and flow diagram of the system life-cycle view for this model is shown in Figure D.1. The levels in this diagram represent an amount of software functionality as it passes through various stages of the software development process. Each level equation is expressed in units of function points where a function point represents an amount of software functionality in some state (e.g., the level "Code To Do" represents the function points that have been designed but not yet coded at any given point in time). The flow equations in this model represent the team's productivity at performing software develop-

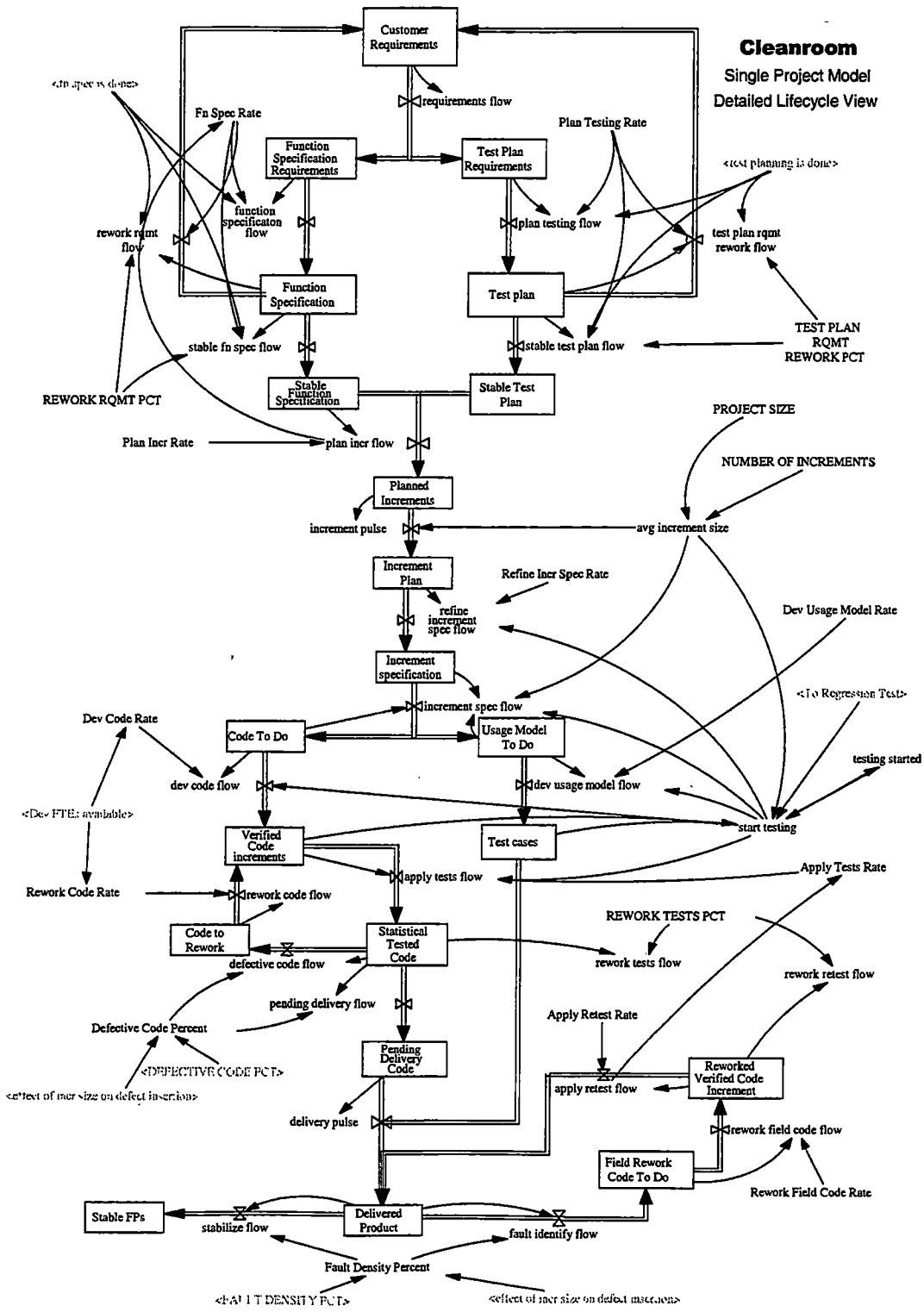


Figure D.1: Life-cycle view of simulation model

ment tasks in units of function points per week. The Customer Requirements level is initialized with the size of the project to be developed as given by PROJECT SIZE. As time progresses from one week to the next, this amount of functionality flows through the model with various tasks performed on the required software functionality at each flow valve.

The primary results we are concerned with are costs and schedule. The project is considered complete when the full amount of stable functionality is delivered to the customer. The variable "Project Completion Time" gives the week in which this milestone is achieved. Total costs for the project is given by the level Total Effort Hours as shown in Figure D.2. Like most cost estimation models, this one computes costs in terms of effort hours which are easily converted to monetary units. A unit cost factor is computed for each task in effort hours per function point. The unit cost factor is multiplied by the work flow for the task to give the effort cost rate in effort hours per week.

Unit costs are based on the productivity for a given task as shown by the example causes tree in Figure D.3. The unit costs for any task is given by

$$\text{<task> Unit Cost} = \frac{\text{HOURS PER PERSON WEEK}}{\text{<task> Productivity}}$$

where, HOURS PER PERSON WEEK is a constant that is set to the default value of 22. This constant takes into account vacations, holidays, training, and other non-productive time averaged over the entire year.

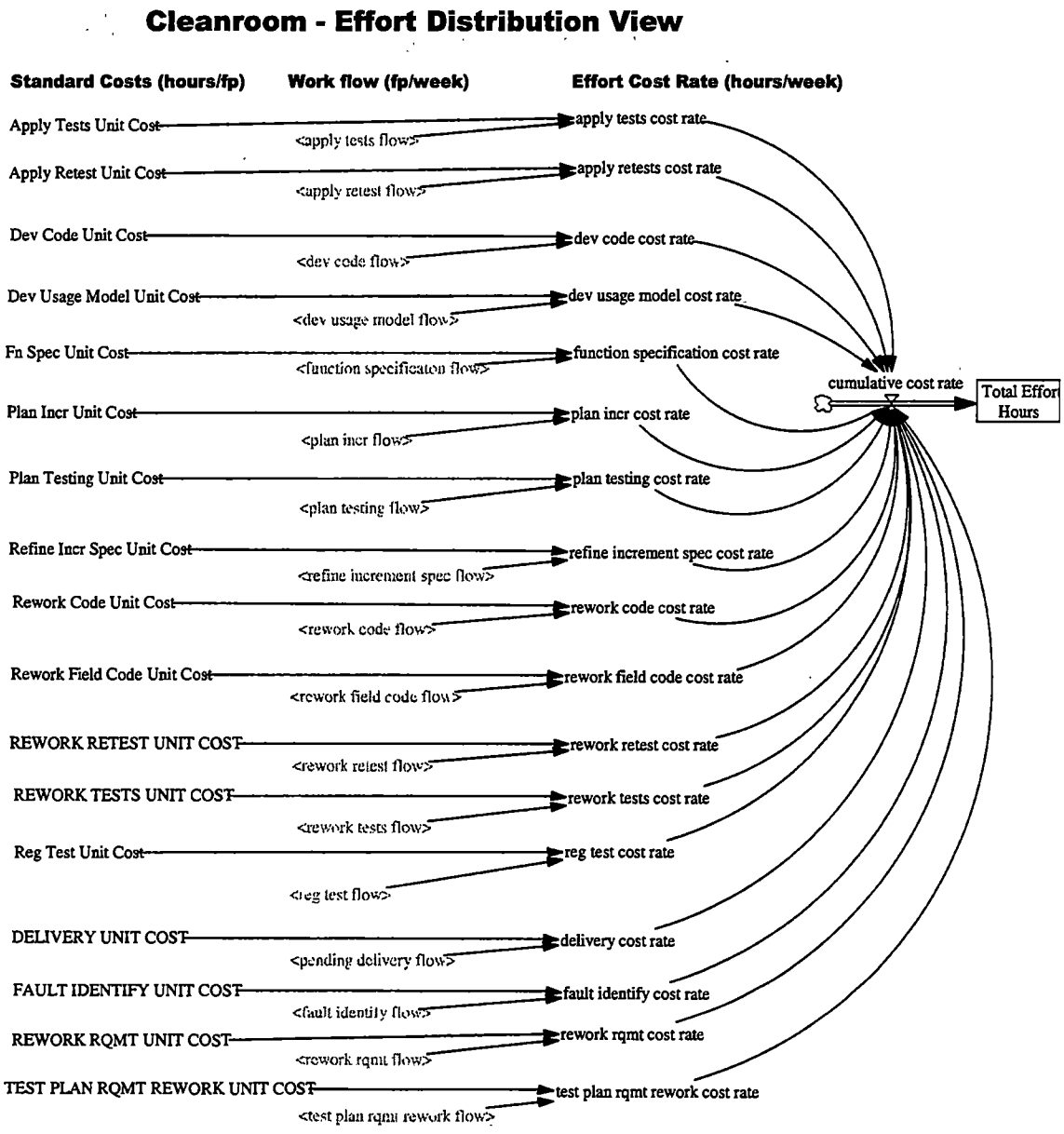


Figure D.2: Computing Total Effort Hours

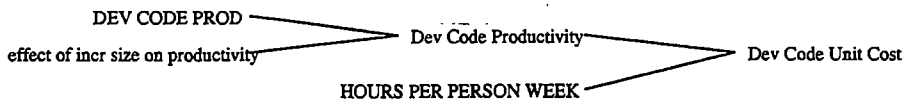


Figure D.3: Example causes tree for a task unit cost

The team's potential productivity rate (<task> rate) is computed from the number of staff resources available (<phase> FTEs AVAILABLE) for a task times the person productivity for a task (<task> Productivity). There are three types of resources we estimate for this model corresponding to three phases of the development life-cycle: Specifiers, Developers, and Testers. These constants are set by using the phase distribution tables for the COCOMO model.[1]

The model adjust allocations of these resources so that competing task demands for the same resources has the correct impacts on costs and schedule.

D.2.1 Model Boundary

The model assumes that the effort begins after requirements have been generally defined and excludes feasibility and requirements efforts. Also, the model excludes indirect activities such as user training, documentation, management, and support activities. Since the COCOMO model includes indirect activities, we reduce the COCOMO values where needed to account for out of scope activities.

1. See Table 6-8 on page 90 of [9].

D.3 Supporting Software

The model was developed using the Vensim simulation environment from Ventana Systems. This simulation tool provides a dynamic link library (DLL) of routines for controlling the model from an external program. An ActiveX DLL COM object was created in order to control the model and simulations from the CBASPI tool. Also, a separate COM object was created for performing the COCOMO estimations of productivity constants and staff allocations.

Appendix E

Software Process Improvement Templates

This dissertation work resulted in templates for three uses of Emerald and for four Cleanroom technologies. This appendix provides listings of the Emerald template for targeted defect reduction and of the Cleanroom template for statistical testing.

E.1 Emerald Template for Targeted Defect Reduction

SPI Template Report

Software Process Improvement	SPI Type
Emerald Targeted Defect Reduction	Emerald Targeted QA

Description Using Emerald to target defect reduction efforts. Defect reduction includes activities to prevent defects from occurring as well as activities to find and remove defects. The primary benefits from this use of Emerald are more efficient resource allocation and a gain from early defect detection.

Cost-Benefit Effects

Costs

Implementation->Tools and information systems

Cost to acquire product or tool TANGIBLE

COSQ Cat: Prevention **BSC Cat:** Internal/Business Process

Cost to make or buy tool or system.

Equations:

start	end yr	formula	label
1	1	=-tool_cost	

Tool / system maintenance costs TANGIBLE

COSQ Cat: Prevention **BSC Cat:** Internal/Business Process

Cost for maintenance of tool or service.

Equations:

start	end yr	formula	label
2		=-tool_maint_cost	

Emerald Targeted Defect Reduction

Implementation->Training

Personnel time in training TANGIBLE

COSQ Cat: Prevention **BSC Cat:** Learning and Growth

Cost for the time personnel will spend in training.

Equations:

start	end yr	formula	label
1		=-num_trainees * training_hrs_per_trainee * loaded_labor_rate	

Implementation->Use and operations

Operations support TANGIBLE

Equations:

start	end yr	formula	label
1		=-tool_admin_hrs_per_wk* Loaded_labor_rate * Weeks_per_year	

Data analysis, summarization and reporting TANGIBLE

COSQ Cat: Prevention **BSC Cat:** Internal/Business Process

Additional time each week for reviewing and analyzing Emerald information.

Equations:

start	end yr	formula	label
1		=-use_cost_yr	

Benefits

Customer / Market Impact->Revenue Impact

Customer satisfaction SEMITANGIBLE

BSC Cat: Customer

Emerald improves productivity, cycle time, and quality of the software product. These are characteristics that customers value and that helps to earn their repeat business and attract new customers.

Equations:

start	end yr	formula	label
1		=SPI_add_business_yr + SPI_repeat_business_yr - baseline_repeat_business_yr	

Emerald Targeted Defect Reduction

Cycle Time Reduction SEMITANGIBLE

COSQ Cat: Internal failure **BSC Cat:** Customer

Emerald reduces schedule because of better utilization of development, inspection, and testing resources and by reducing defects and rework from formal testing.

Equations:

start	end yr	formula	label
1		=cycle_reduct_value	Value of cycle time reduction

Production Cost Impact->Management->Decision Support

Improve managing process change SEMITANGIBLE

COSQ Cat: Prevention **BSC Cat:** Learning and Growth

Provides objective data to quantify and support change decisions.

Production Cost Impact->Management->Project Oversight and Tracking

Keeping project on schedule SEMITANGIBLE

Schedule slippage leads to inefficient use of resources and increases overall cost of the project. Metrics inform management decisions early and often to enable corrective actions

Production Cost Impact->Management->Project Planning

Improved project planning and estimating SEMITANGIBLE

Improved resource allocation decisions and staffing assignments SEMITANGIBLE

Quality Effect->External Failure Costs

Enhanced understanding of field problems SEMITANGIBLE

Emerald Targeted Defect Reduction

Quality Effect->Quality Appraisal->Inspection, Validation and Verification->Inspection Time Savings

Code Inspection Time Savings TANGIBLE

COSQ Cat: Appraisal **BSC Cat:** Internal/Business Process

By using Emerald to improve resource allocation an organization can reduce the overall effort required for inspection, testing and development activities.

Equations:

start	end yr	formula	label
1		=baseline_insp_cost - SPI_insp_cost	

Model Parameters

parameter name	user prompt	data type	len
hide lock formula		low val hi val default	
avg_effort_insp	Average effort hours per inspection	Single	5
☐ ☐ 3.5		3.5	
avg_loc_insp	Average lines of code per baseline inspection	Long Integer	5
☐ ☐ 250		250	
base_pct_of_code_reviewed	Baseline percent of code reviewed each year	Percent	5
☐ ☐ 80%		80%	
The percentage of code subject to code review each year			
baseline_insp_cost	Baseline inspection cost	Currency	8
☑ ☐ =reviewed_loc * (avg_effort_insp / avg_loc_insp) * loaded_labor_rate			
baseline_repeat_business_yr	Baseline repeat business per year	Currency	10
☐ ☐			
How much net repeat business occurs each year.			
churned_loc	Churned lines of code	Long Integer	8
☑ ☑ =current_size_LOC * code_churn_pct			

Emerald Targeted Defect Reduction

code_churn_pct	Percentage of code "in play"?	Percent			3
☑ ☑		0%	100%		
	The active code that is subject to active maintenance or modifications. Populated automatically from language form.				
current_size_LOC	Current code base size in Lines of Code	Long Integer			7
☑ ☑					
	Calculated based on other values. Automatically populated by Baseline Language form.				
cycle_day_value	Value of each cycle day saved	Currency			10
☐ ☐	$1,000			$1,00	
	The value of a reduced cycle day. The present value of saving a cycle day in terms of increased sales, awards attained, penalties avoided. The value depends on the software product and its marketplace.				
cycle_reduct_value	Annual value of cycle reduction	Currency			5
☑ ☐	=proj_days_saved_yr * cycle_day_value				
	The present value of cycle reduction accrued for the current year				
dev_staff_size	Size of development staff	Long Integer			5
☑ ☐					
	List the number of development personnel involved in the software projects being developed in this environment. Do not include managers, or QA/testing personne				
grn_effort	Average effort to inspect a low risk ("green") module	Single			4
☐ ☐	2			2	
	Average effort hours to inspect a low risk green module.				
grn_loc_insp	Average lines of code covered per inspection of a low risk module	Long Integer			5
☐ ☐	400			400	
loaded_labor_rate	Loaded hourly labor rate	Currency			6
☐ ☐	$60	40	200	80	
	Average loaded hourly rate for development and testing personnel.				
new_loc	New lines of code	Long Integer			8
☑ ☑					

Emerald Targeted Defect Reduction

New_or_changed_LOC	New or changed lines of code	Long Integer	7
☑ ☑	=New_LOC + churned_LOC		
	Calculated by language form.		
num_projects	Number of projects developed within one year	Integer	4
☐ ☐			
	Specify the number of active software projects are being developed or maintained in this environment.		
num_tool_administrators	How many administrators needed for the tool?	Long Integer	5
☐ ☐	1	1	
	The number of administrators needed for the tool.		
num_tool_users	Number of tool users	Long Integer	4
☐ ☐	5 0 1000	5	
	The number of potential users who would need access to an SPI tool.		
num_trainees	Number of people to be trained	Long Integer	4
☐ ☐	=num_tool_users + num_tool_administrators		
	The number of people to be trained in the SPI technology or tool.		
prerelease_savings	Pre-release labor cost savings	Currency	8
☑ ☐	=(baseline_insp_cost - SPI_insp_cost) +SPI_InternalFailureSavings		
proj_days_saved_yr	Project days saved per year	Single	5
☐ ☐	= staff_days_saved_yr / staff_day_to_proj_day		
	An estimate of the number of project days saved per year due to the SPI.		
red_effort	Average effort hours to inspect a high risk ("red") module	Single	5
☐ ☐	4	4	
red_loc_insp	Average lines of code reviewed per inspection of a high risk ("red") module	Long Integer	4
☐ ☐	150	150	
red_ratio	Portion of the reviewed code estimated to be of high risk ("red")	Single	5
☐ ☐	0.2	0.2	

Emerald Targeted Defect Reduction

reviewed_loc	Reviewed lines of code	Long Integer	7
☑ ☐	=New_or_changed_LOC * base_pct_of_code_reviewed		
	The number of new or changed lines of code reviewed each year by the organization.		
SPI_add_business_yr	Additional business per year under SPI scenario	Currency	10
☐ ☐			
	Additional business expected each year under the SPI scenario.		
SPI_insp_cost	Emerald inspection cost	Currency	8
☑ ☐	=reviewed_loc * loaded_labor_rate * (red_ratio * (red_effort / red_loc_insp) + (1 - red_ratio) * (grn_effort / grn_loc_insp))		
	The estimated cost to review code when using Emerald to focus inspection effort:		
SPI_InternalFailureSavings	Internal failure cost savings	Currency	8
☑ ☑			
	Savings in internal rework by reducing internal failures. This variable is set by the Gain From Early Defect Detection model built into the tool. (See the est_project_impact_early_detection function).		
SPI_repeat_business_yr	Annual repeat business under SPI scenario	Currency	10
☐ ☐			
	Repeat business expected each year under the SPI scenario.		
staff_day_to_proj_day	Staff day to project day	Single	5
☐ ☐	= dev_staff_size / num_projects		
	The number of staff days required to reduce the cycle time of the project by one day. Suggested value is the average number of developers per project. (The default formula below estimates the average project size as the total staff divided by the number of projects.)		
staff_days_saved_yr	Staff days saved per year	Single	10
☑ ☐	=staff_hrs_saved_yr / staff_proj_hrs_day		
	The average number of staff days saved per year from targeted defect reduction.		
staff_hrs_saved_yr	Staff hours saved per year	Single	8
☑ ☐	=prerelease_savings / loaded_labor_rate		
	Annual staff hour savings from targeted defect reduction.		

Emerald Targeted Defect Reduction

staff_proj_hrs_day Staff hours per day on project Single 3

☑ ☐ 6 1 8 6

The average amount of time a staff person spends on project activities.

tool_admin_hrs_per_wk Tool administration hours per week Single 4

☐ ☐ 4 4

Estimated hours per week for tool system administration tasks such as maintaining users, monitoring batch jobs, usage, and reconfiguring the system a needed.

tool_client_license_cost Cost of client license for SPI tool Currency 7

☐ ☐ $4,000 0 50000

The cost of each client license for an SPI tool.

tool_cost Initial cost of tools to facilitate SPI Currency 7

☑ ☐ =tool_client_license_cost*num_tool_users+tool_erver_license_cost

This is the initial cost to purchase client and server licenses for the tool.

tool_maint_cost Tool maintenance cost Currency 7

☑ ☐ =tool_cost*tool_maint_pct

The annual cost of tool maintenance

tool_maint_pct Annual tool maintenance percentage Percent 4

☐ ☐ 18% 0.00 0.50 0.05

The annual cost of a tool maintenance contract as a percentage of the initial purchase.

tool_server_license_cost Server license cost Currency 5

☐ ☐ $130,000

The cost of a server license for the SPI tool.

training_hrs_per_trainee Training hours per trainee Single 5

☐ ☐ 4

The number of hours each trainee would need to spend in training.

use_cost_yr The annual cost of using the tool Currency 8

☑ ☑ =use_hrs_wk * loaded_labor_rate * weeks_per_year

The annual cost of taking additional time to review and analyze metrics and decision support information.

Emerald Targeted Defect Reduction

use_hrs_wk — Effort hours spent reviewing metrics information each week — Currency — 8

☐ ☐ 4 4

An estimate of the additional hours per week management and users will spend reviewing and analyzing metrics and decision support information.

weeks_per_year — Weeks per year — Integer — 2

☑ ☐ 50 1 52 50

Process Impact

action	appraisal step	seq	description	rem. eff.	impact:
	Requirements inspections	1600	A formal review of the requirements specification intended to find inconsistent, invalid or missing requirements.	35.00%	0.00%
	Informal design reviews	2500	An informal review of the design to verify that it meets requirements.	30.00%	0.00%
	Formal design inspections	2600	A formal review of the design to verify that the system design meets the intended requirements.	65.00%	0.00%
	Informal code reviews	3500	Informal review of code	30.00%	10.00%
	Formal code inspection	3600	Formal code inspection	65.00%	10.00%
	Functional verification	3700	Rigorous verification of software products	70.00%	10.00%
	Subroutine testing	3800	Testing of subroutines. Usually informal testing of a subroutine by the developer to ensure it compiles and performs properly.	19.00%	0.00%
	Unit testing	5150	The execution of a complete module or small program that will normally range from about 100 to 1000 source code statements, or roughly 1 to 10 function points.	30.00%	0.00%
	New function testing	5200	Aimed at validating new features that are added to a new or modified program and to check for intermodule interfaces. This form of testing is also called "component testing." Often combines the work of multiple programmers in to a component of a larger system. New function testing is normally supported by formal test plans and planned test cases.	30.00%	5.00%
	Regression testing	5260	Test code against accidentally damaging an existing feature from adding a new feature. Also, insure that prior known bugs are actually removed from code modifications intended to correct the problems.	23.00%	0.00%
	Integration testing	5300	Testing of a number of modules or programs that have come together to comprise an integrated software package.	35.00%	5.00%

				Emerald Targeted Defect Reduction	
	Performance testing	5350	Testing aimed at judging whether or not an application can meet specified performance requirements.	23.00%	0.00%
Modify	System testing	5370	System testing of the full application. This is often the last form of internal testing before customers get involved with field (or beta) testing.	36.00%	10.00%
Modify	Platform testing	5400	A specialized form of testing found among companies whose software operates on different hardware platforms under different operating systems.	24.00%	5.00%
	Independent testing	5420	Testing performed by a separate company or organization from the one that built the application.	31.00%	0.00%
	Lab testing	5440	A special form of beta testing where a company has a laboratory where clients can test out both hardware and software prior to having the equipment installed on their own premises.	38.00%	0.00%
	Security testing	5450	Testing to ensure software security requirements have been satisfied.	16.00%	0.00%
	Stress or capacity testing	5470	A form of testing that verifies the system will perform properly for large transaction or data loads.	29.00%	0.00%
	Random testing	5500	Testing involving a random selection of test cases in order to make statistical inferences about the quality of the software.	30.00%	0.00%
	Usage based testing	5520	Statistical testing that models the expected usage patterns of the software in order to make inferences about the reliability of the software in field use.	15.00%	0.00%
	Usability testing	5550	Testing often performed by actual end users who utlize the system under controlled conditions (often video taped) where the user's actions can be analyzed.	14.00%	0.00%
	Viral protection testing	5600	Testing aimed at ensuring software viruses have not been introduced into the product.	16.00%	0.00%
	Year 2000 testing	5650	A specialized form of testing aimed at identifying the presence of Year 2000 problems.	16.00%	0.00%
	Euro conversion testing	5670	A specialized form of testing for ensuring the Euro currency will be handled properly.	16.00%	0.00%
	International testing	5690	Testing to ensure versions of software work correctly for an international audience; test for language, currency conversions, all strings stored in separate configurable files, etc.	16.00%	0.00%
	Customer acceptance testing	5700	Customers test the software prior to accepting it.	15.00%	0.00%
	Formal Qualification Testing (FQT)	5710	Set of formal tests.	1.00%	0.00%

			Emerald Targeted Defect Reduction	
Low volume Beta test (< 10 clients)	5750	External field testing performed by less than 10 customers.	30.00%	0.00%
Flight test	5752	Flight testing.	30.00%	0.00%
Med volume Beta test (10-1000 clients)	5770	Field testing performed by 10 to 1000 customers.	50.00%	0.00%
High volume Beta test (> 1000 clients)	5790	Extensive field testing performed by over 1000 clients.	75.00%	0.00%
Field use	6500	Application is deployed and customers find errors during production use of the software.	90.00%	0.00%

Emerald Targeted QA's relationships with other SPIs:

Relation	SPI	Comments
is enhanced by	Metrics and Decision Support	

SPI References

Emerald Targeted QA

Ref Id	Citation
[Hudepohl 1996]	J. P. Hudepohl, et. Al., Emerald: Software Metrics and Models on the Desktop, IEEE Software, 56-60, Sept. 1996
[Hudepohl 1997]	Hudepohl, J. P., Network Software Reliability and Quality, The Froehlich/Kent Encyclopedia of Telecommunications, 281-314, 1997

E.2 Cleanroom Template for Statistical Testing

SPI Template Report

Software Process Improvement

Statistical Testing

Description Statistical testing involves developing a usage model (e.g., in the form of a Markov Chain) to represent all possible ways the system can be used. Statistical testing provides improved decision support of when testing is completed and can be safely released.

Cost-Benefit Effects

Costs

Implementation->Tools and information systems->Cost to acquire product or tool

Purchase of Usage Modeling tool TANGIBLE

COSQ Cat: Prevention **BSC Cat:** Learning and Growth

Specialized tools for usage modeling greatly facilitate the process.

Equations:

start	end yr	formula	label
1		=-tool_cost	

Implementation->Tools and information systems->Tool / system maintenance costs

Annual maintenance fees for Usage Modeling tool TANGIBLE

COSQ Cat: Prevention **BSC Cat:** Learning and Growth

Tools typically require annual fees to cover maintenance, access to a help desk and new releases.

Equations:

start	end yr	formula	label
1		=-tool_maint_cost	

Statistical Testing

Implementation->Training

Personnel time in training TANGIBLE

COSQ Cat: Prevention **BSC Cat:** Learning and Growth

Personnel who currently perform testing can be re-trained to perform usage modeling and statistical testing. An estimate of the number of usage modelers can be estimated by dividing the total usage model development hours required each year by 150.

Equations:

start	end yr	formula	label
1	1	=-num_usage_modelers * training_hrs_per_trainee * loaded_labor_rate	

Implementation->Training->Costs to purchase training

Training class TANGIBLE

COSQ Cat: Prevention **BSC Cat:** Learning and Growth

The cost to purchase training for those who will perform usage modeling.

Equations:

start	end yr	formula	label
1	1	=-num_usage_modelers * training_cost_per_modeler	

Cost of consultants and coaching TANGIBLE

COSQ Cat: Prevention **BSC Cat:** Learning and Growth

Consultants to provide advice and coaching to help teams implement a new technology or process improvement.

Equations:

start	end yr	formula	label
1	1	=-consulting_hrs * (1-consult_direct_pct) * consult_hr_rate	

Statistical Testing

Quality Effect->Quality Appraisal->Testing

Time to analyze, develop and maintain usage model TANGIBLE

COSQ Cat: Prevention **BSC Cat:** Customer

The process of developing the high level usage specification helps to define functional specifications and requirements. If functional specifications already exist it is easier to develop a user specification because decisions have already been made about how the application will flow. Conversely, once a usage specification exists, it is easier to complete a functional specification. The usage specification process forces customers to specify how the application will flow. The additional effort for creating usage models is in documenting the usage states in the Markov chain format and estimating transition probabilities for each mode of use.

Equations:

start	end yr	formula	label
1		=-usage_model_dev_hrs * loaded_labor_rate	

Time to interface usage test generator to testing tool SEMITANGIBLE

COSQ Cat: Appraisal **BSC Cat:** Internal/Business Process

To realize the efficiencies from automated testing there may be some effort involved to interface the output of the usage model to a specific testing tool or testing environment. The capabilities of usage testing tools should provide interfaces to most common automated testing environments. However, there may be some work required to tune output for specific environments.

Equations:

start	end yr	formula	label
1	1	=-interface_tool_hrs * loaded_labor_rate	

Benefits

Customer / Market Impact->Revenue Impact

Customer retention SEMITANGIBLE

BSC Cat: Customer

Improved field quality leads to higher customer satisfaction which can help retain existing customers and attract new customers.

Equations:

start	end yr	formula	label
1		=SPI_add_business_yr + SPI_repeat_business_yr - baseline_repeat_business_yr	

Statistical Testing

Reduced cycle time value SEMITANGIBLE

BSC Cat: Customer

The test cycle time is reduced by the same factors that reduce testing cost. The usage modeling and test case generation can be performed in parallel with the software development.

Equations:

start	end yr	formula	label
1		=proj_days_saved_yr * cycle_day_value	

Production Cost Impact->Documentation->Requirements

Reduced requirements creep cost TANGIBLE

COSQ Cat: Prevention **BSC Cat:** Customer

Usage specification and modeling mitigate the risk of requirements creep later in the life cycle. Jones states that "defect rates associated with new features added during mid-development are about 50% greater than those of the artifacts associated with the original requirements. Defect removal efficiency levels are depressed as well, sometimes by more than 15%."

Equations:

start	end yr	formula	label
1		=reduce_rqmts_creep_savings	

Production Cost Impact->Maintenance

Reduced maintenance cost TANGIBLE

COSQ Cat: External Failure **BSC Cat:** Customer

Statistical testing contributes to improved field quality of the software product. Defects are efficiently removed for a given test budget resulting in higher quality once the testing budget has been exhausted. Also, the quantitative analysis and improved prediction of field quality improves process control and release decisions. Higher field quality results in lower field support costs and higher customer satisfaction.

Equations:

start	end yr	formula	label
1		=maint_cost_savings	

Statistical Testing

Production Cost Impact->Management->Decision Support

Improved support for release decisions SEMITANGIBLE

COSQ Cat: External Failure **BSC Cat:** Internal/Business Process

Quantitative data from the testing process can help managers make better estimates and decisions on when to stop testing and release the software.

Equations:

start	end yr	formula	label
1		=SPI_decision_savings_yr	

Production Cost Impact->Management->Project Planning

Improved resource and schedule planning for testing phase SEMITANGIBLE

COSQ Cat: Appraisal **BSC Cat:** Internal/Business Process

The usage model can provide data for effort, schedule, and cost projections that will be required for the testing effort.

Quality Effect->Quality Appraisal->Inspection, Validation and Verification

Validation of requirements SEMITANGIBLE

COSQ Cat: Prevention **BSC Cat:** Customer

Better validation of requirements results in better requirements stability less growth of requirements later on in the lifecycle.

Quality Effect->Quality Appraisal->Testing

Targeted more effective testing efforts and test automation efforts SEMITANGIBLE

COSQ Cat: Appraisal **BSC Cat:** Internal/Business Process

Statistical testing and being able to generate random test cases from a usage model so that the test budget is spent in a way that maximizes the increase in operational reliability.

Automated test case generation SEMITANGIBLE

COSQ Cat: Appraisal **BSC Cat:** Internal/Business Process

Once a usage model is built it can be used to generate random test cases from the usage model. Test cases are generated by taking random walks through the model and generating program input for each arc transition. Automated test case generation saves human effort to manually generate the same number of test cases. Also, the test cases can be automatically executed saving additional effort.

Equations:

start	end yr	formula	label
1		=auto_test_gen_savings1	

Statistical Testing

Effective, efficient testing SEMITANGIBLE

COSQ Cat: Appraisal **BSC Cat:** Internal/Business Process

The generated test cases will test paths in the program with the same frequency distribution as is expected during operational use. Faults on frequently traversed paths have the highest probability of causing a failure in the field. Thus "the test budget is spent in a way that maximizes the increase in operational reliability from testing." Note that this improvement should not include efficiencies due to automatic test case generation.

Equations:

start	end yr	formula	label
1		=test_cost_yr * test_improve_pct	

Model Parameters

parameter name	user prompt	data type			len
hide lock	formula	low val	hi val	default	
auto_test_gen_savings1	Automatic test generation savings 1	Currency			8
☑ ☐	=avg_test_case_prep_hrs * loaded_labor_rate *SPI_num_test_cases_yr				
	One approach to estimating the amount of manual effort that is displaced from having the ability to automatically generate test cases. (See auto_test_gen_savings2 for a second approach.)				
avg_test_case_prep_hrs	Average time to prepare a test case in hours	Single			5
☐ ☐	0.5			0.5	
base_decision1_state1_prob	Baseline probability of decision 1 and state 1	Percent			5
☑ ☑	=state_1_prob *baseline_correct_prediction				
	The probability that decision 1 is selected given that state 1 is the true state of nature.				
base_decision1_state2_prob	Baseline probability of decision 1 and state 2	Percent			5
☑ ☑	=(1 - state_1_prob) * (1 - baseline_correct_prediction)				
	The probability that decision 1 is selected given that state 1 is the true state of nature.				
base_decision2_state1_prob	Baseline probability of decision 2 and state 1	Percent			5
☑ ☑	= state_1_prob * (1 - baseline_correct_prediction)				
	Probability of decision 2 is made and state 1 is true state of nature				

Statistical Testing

base_decision2_state2_pr Baseline probability of decision 2 and state 2 Percent 5
b

☑ ☑ =(1 - state_1_prob) * baseline_correct_predictior

Probability that decision 2 is chosen and state 2 is true state of nature.

baseline_correct_predictior Percentage of time correct decision made under baseline scenario Percent 5

☑ ☑ 60%

Percentage of time the organization correctly make the correct choice under the baseline scenario. This variable must always point to a specific variable for the baseline.

baseline_cost_per_decisio Baseline cost per decision Currency 8
n

☑ ☑ =payoff_decision1_state1 *
base_decision1_state1_prob +
payoff_decision1_state2 *
base_decision1_state2_prob +
payoff_decision2_state1 *
base_decision2_state1_prob +
payoff_decision2_state2 *
base_decision2_state2_prob

baseline_repeat_business Baseline repeat business per year Currency 10
yr

☐ ☐

How much net repeat business occurs each year.

churned_loc Churned lines of code Long Integer 8

☑ ☑ =current_size_LOC * code_churn_pct

code_churn_pct Percentage of code "in play"? Percent 3

☑ ☑ 0% 100%

The active code that is subject to active maintenance or modifications. Populated automatically from language form.

code_creep_reduce Coding phase creep reduction Percent 5

☑ ☐ 10% 10%

Percentage of requirements creep eliminated from coding phase from the SPI.

code_high_quality_prob Probability that code is high quality at time of release decision Percent 5

☐ ☐ 50% 50%

At the time a decision is made to release the software, the true quality of the code is unknown. This represents the probability that the code is of acceptable release quality at the time of the decision.

Statistical Testing

code_phase_budget_pct — Percent of budget spent during coding phase — Percent — 5

☐ ☐ 21% — 21%

code_phase_creep_mth — Percent of new requirements per month of coding phase — Percent — 5

☑ ☐ 1.25% — 1.25%

code_rel_cost_fp — Relative cost per function point for requirements introduced during coding — Single — 5

☑ ☐ 1.5 — 1.5

The relative cost per function point for requirements introduced during coding.

consult_direct_pct — Consulting direct contribution percent — Percent — 4

☐ ☐ 0.90 — 0 — 1 — 0.90

The percentage of consulting time that directly contributes to product development.

consult_hr_rate — Consultant hourly rate — Currency — 5

☐ ☐ 80 — $20 — $500 — $80

The hourly cost per consultant that is needed to implement the SPI.

consulting_hrs — Consulting hours to implement SPI — Long Integer — 5

☐ ☐ 80

The number of consulting hours that will be needed to implement the SPI.

current_size_LOC — Current code base size in Lines of Code — Long Integer — 7

☑ ☑

Calculated based on other values. Automatically populated by Baseline Language form.

cycle_day_value — Value of each cycle day saved — Currency — 10

☐ ☐ 1000 — $1,00

The value of a reduced cycle day. The present value of saving a cycle day in terms of increased sales, awards attained, penalties avoided. The value depends on the software product and its marketplace.

decisions_per_year — Decisions made per year — Long Integer — 4

☑ ☑ 5

The number of decisions made each year.

design_creep_reduce — Percent of requirements creep reduced in design phase — Percent — 5

☑ ☐ 10% — 10%

Statistical Testing

design_phase_budget_pct — Percent of budget spent during design phase — Percent — 5

☐ ☐ 23% 23%

The percentage of the development budget spent during design phase.

design_phase_creep_mth — Percent of requirements introduced during design phase per month — Percent — 5

☑ ☐ 3% 3%

design_rel_cost_fp — Relative cost per function point for requirements introduced during design — Single — 5

☑ ☐ 1.25 1.25

The relative cost per function point for a requirement introduced during this phase

interface_tool_hrs — Effort hours to adapt tool to particular environment — Single — 5

☐ ☐ 80

The hours involved to adapt and implement a tool for a particular development environment.

KLOCS_per_year — New or changed KLOCS per year — Single — 5

☑ ☑ =New_or_changed_LOC / 1000

loaded_labor_rate — Loaded hourly labor rate — Currency — 6

☐ ☐ $80 40 200 80

Average loaded hourly rate for development and testing personnel.

maint_cost_savings — Maintenance cost savings — Currency — 8

☑ ☐ =maint_cost_yr * maint_reduct_pct

A reduction in software maintenance cost due to the SPI.

maint_cost_yr — Maintenance cost per year — Currency — 8

☐ ☐ =maint_effort_pct * software_budget_per_yr

The annual cost of software maintenance.

maint_effort_pct — Percent of budget spent on software maintenance — Percent — 5

☐ ☐ 20%

The percentage of the software budget spent on maintenance of existing systems

maint_reduct_pct — Percent reduction in maintenance cost — Percent — 5

☐ ☐ 10%

A percentage reduction in maintenance cost due to implementation of the SPI.

Statistical Testing

new_loc	New lines of code		Long Integer	8
☑ ☑				
New_or_changed_LOC	New or changed lines of code		Long Integer	7
☑ ☑	=New_LOC + churned_LOC			
	Calculated by language form.			
num_tool_users	Number of tool users		Long Integer	4
☐ ☐	=num_usage_modelers	0	1000	5
	The number of potential users who would need access to an SPI tool.			
num_usage_modelers	Number of usage modelers		Long Integer	4
☐ ☐				
	The number of people that will develop statistical usage models.			
payoff_decision1_state1	Payoff for decision 1 when state 1 occurs		Currency	10
☑ ☑	=payoff_release_good_quality			
payoff_decision1_state2	Payoff for decision 1 when state 2 occurs		Currency	8
☑ ☑	=payoff_release_poor_quality			
payoff_decision2_state1	Payoff for decision 2 when state 1 occurs		Currency	8
☑ ☑	=payoff_test_good_quality			
	The payoff for decision 2 when state 1 occurs			
payoff_decision2_state2	Payoff for decision 2 when state 2 occurs		Currency	8
☑ ☑	=payoff_test_poor_quality			
	The payoff for decision 2 when state 2 occurs.			
payoff_release_good_quality	Payoff for releasing good quality code on-time		Currency	8
☐ ☐				
payoff_release_poor_quality	Payoff for releasing poor quality code		Currency	10
☐ ☐				
	What is the financial consequence of releasing poor quality code.			
payoff_test_good_quality	Payoff for delaying release of good quality code		Currency	10
☐ ☐				
	What is the financial consequence of delaying the release in order to test code that is already of acceptable quality?			

Statistical Testing

payoff_test_poor_quality — Payoff for delaying the release of poor quality code for continued testing — Currency — 8

☐ ☐

What is the payoff for delaying the release of poor quality code for continued testing?

proj_days_saved_yr — Project days saved per year — Single — 5

☐ ☐ 20

An estimate of the number of project days saved per year due to the SPI.

reduce_rqmts_creep_savings — Savings from reduction of requirements creep — Currency — 10

☑ ☑ =software_budget_per_yr * 12 * (design_phase_budget_pct * design_rel_cost_fp * design_phase_creep_mth * design_creep_reduce + code_phase_budget_pct * code_rel_cost_fp * code_phase_creep_mth * code_creep_reduce + test_phase_budget_pct * test_rel_cost_fp * test_phase_creep_mth * test_creep_reduce)

software_budget_per_yr — Software development budget per year — Currency — 7

☐ ☐

Enter the estimated development budget per year. This can be estimated from the average fully loaded developer salary times the size of the staff.

SPI_add_business_yr — Additional business per year under SPI scenario — Currency — 10

☐ ☐

Additional business expected each year under the SPI scenario.

SPI_correct_prediction — Percentage of time correct decision will be made under SPI scenario — Percent — 5

☑ ☑ 95%

The right choices are considered to be decision 1 when state 1 is the true state of nature, and decision 2 when state 2 is the true state of nature.

SPI_cost_per_decision — SPI cost per decision — Currency — 8

☑ ☑ =payoff_decision1_state1 * SPI_decision1_state1_prob + payoff_decision1_state2 * SPI_decision1_state2_prob + payoff_decision2_state1 * SPI_decision2_state1_prob + payoff_decision2_state2 * SPI_decision2_state2_prob

			Statistical Testing
SPI_decision_savings_yr	Annual SPI decision savings	Currency	8
☑ ☐	=(SPI_cost_per_decision - baseline_cost_per_decision) * decisions_per_year		
SPI_decision1_state1_prot	SPI probability of decision 1 and state 1	Percent	5
☑ ☑	=state_1_prob * SPI_correct_prediction		
SPI_decision1_state2_prot	SPI probability of decision 1 and state 2	Percent	5
☑ ☑	=(1 - state_1_prob) * (1 - SPI_correct_prediction)		
SPI_decision2_state1_prot	SPI probability of decision 2 and state 1	Percent	5
☑ ☑	=state_1_prob * (1 - SPI_correct_prediction)		
SPI_decision2_state2_prot	SPI probability of decision 2 and state 2	Percent	5
☑ ☑	=(1 - state_1_prob) * SPI_correct_prediction		
SPI_KLOC_yr	How many KLOCs are subject to the SPI each year	Double	5
☐ ☐	=new_loc/1000		
	This value is the estimated Kilo- lines of source code (non-comment, non-blank lines containing source code) where the software process improvement will be applied each year.		
SPI_num_test_cases_yr	Number of test cases applicable to the SPI each year	Long Integer	4
☐ ☐			
	The number of test cases applicable to the SPI each year.		
SPI_repeat_business_yr	Annual repeat business under SPI scenario	Currency	10
☐ ☐			
	Repeat business expected each year under the SPI scenario.		
state_1_prob	State 1 probability	Percent	5
☑ ☑	=code_high_quality_prob	50%	
	The percentage of the time that state 1 is the true state of nature. Note that the probabilty of state 2 is (1 - state_1_prob).		
test_cost_yr	Testing cost per year	Currency	8
☐ ☐	=test_effort_pct * software_budget_per_yr		
	Annual budget spent on testing each year.		

Statistical Testing

test_creep_reduce — Test phase creep reduction — Percent — 5
☐ ☐ 10% — 10%

test_effort_pct — Percent of budget spent on testing — Percent — 5
☐ ☐ 15% — 15%
The percent of the budget spent on testing.

test_gen_pct — Percentage of testing budget that would be displaced by SPI — Percent — 5
☐ ☐ 5% — 5%
The portion of the annual development budget that is currently spent on manual test case generation that would be displaced by the SPI.

test_improve_pct — Percent improvement in testing productivity — Percent — 5
☐ ☐ 10%
Percentage improvement in testing productivity as a result of the SPI

test_phase_budget_pct — Percentage of budget spent during testing phase — Percent
☐ ☐ 30% — 30%

test_phase_creep_mth — Percent of new requirements per month of testing phase — Percent — 5
☑ ☐ 0.5% — 0.5%

test_rel_cost_fp — Relative cost per function point for requirements introduced during testing — Single — 5
☐ ☐ 2.5 — 2.5

tool_client_license_cost — Cost of client license for SPI tool — Currency — 7
☐ ☐ $500 — 0 — 50000
The cost of each client license for an SPI tool.

tool_cost — Initial cost of tools to facilitate SPI — Currency — 7
☑ ☐ =tool_client_license_cost * num_tool_users + tool_server_license_cost
This is the initial cost to purchase client and server licenses for the tool.

tool_maint_cost — Tool maintenance cost — Currency — 7
☑ ☐ =tool_cost*tool_maint_pct
The annual cost of tool maintenance

Statistical Testing

tool_maint_pct	Annual tool maintenance percentage	Percent			4
☐ ☐	5%	0.00	0.50	0.05	
	The annual cost of a tool maintenance contract as a percentage of the initial purchase.				
tool_server_license_cost	Server license cost	Currency			5
☐ ☐	$1,000				
	The cost of a server license for the SPI tool.				
training_cost_per_modeler	Training cost per modeler	Currency			5
☑ ☐	$1,500			1500	
	Expected cost of training each trainee.				
training_hrs_per_trainee	Training hours per trainee	Single			5
☐ ☐	40				
	The number of hours each trainee would need to spend in training.				
usage_model_dev_hrs	Annual effort hours to develop usage models	Double			5
☑ ☐	=36 * SPI_KLOC_yr + 15				
	We have low confidence in the default equation as it is based on only a very few data points. More study is needed to establish improved measures.				

Statistical Testing's relationships with other SPIs:

Relation	SPI	Comments
enhances	Sequence Based Specification	The process of developing the high level usage specification helps to define functional specifications and requirements. Once a usage specification exists, it is easier to complete a functional specification. The usage specificatior process forces customers to specify how the application will flow. The additional effort for creating usage models is in documenting the usage states in the Markov chain format and estimating transition probabilities for each mod of use.
enhances	Incremental Development	Provides feedback on process quality by performing statistical testing at the end of each increment.
is component of	Cleanroom Software Engineering	

		Statistical Testing
is enhanced by	Sequence Based Specification	If functional specifications already exist it is easier to develop a user specification because decisions have already been made about how the application will flow. Usage models can be generated from sequenced-based specifications.

SPI References

Statistical Testing

Ref Id	Citation
[Agrawal93]	K. Agrawal, J. Whittager, Experiences in Applying Statistical Testing to a Real-Time, Embedded Software System, Proceedings of the Pacific Northwest Software Quality Conference, 154-170, 1993
[Basili1994]	Basili, Victor, and Scott Green, Software Process Evolution at the SEL, IEEE Software, 58-66, July 1994
[Brewer95]	M. Brewer, P. Fisher, D. Fuhrer, K. Nielsen, Poore, The Application of Cleanroom Software Engineering to the Development of Embedded Control Systems Software, Proceedings of the Second European Industrial Symposium on Cleanroom Software Engineering, , Mar
[Fuhrer 1992]	David Fuhrer, Hailong Mao, J.H. Poore, OS/2 Cleanroom Environment: A Progress Report on a Cleanroom Tools Development Project, Proceedings of the Hawaii International Conference on System Sciences (1992), 449-458, 1992
[Poore1998]	Poore, Jesse H., and Carmen J. Trammell, Application of Statistical Science to Testing and Evaluating Software Intensive Systems, Statistics, Testing, and Defense Acquisition, , 1998
[Tann 1993]	L-G Tann, A Successful Cleanroom Project - OS32, European Industrial Symposium on Cleanroom Software Engineering, 40, 1993
[Whittaker94]	James A. Whittaker, A Case-Study in Software Reliability Measurement, Quality Week '94, 1-17, May 1994

Appendix F

Example Cost-Benefit Analyses

Examples cost-benefit analyses were created which used the templates for the three uses of Emerald and for the four Cleanroom technologies. This appendix presents example CBA reports Emerald targeted defect reduction and for Cleanroom statistical testing.

F.1 Emerald Example for Targeted Defect Reduction

Impact of process improvement on baseline environment

SPI project: Emerald Targeted Defect Red
cost-benefit analysis name: Test CBA of Emerald Targeted Defect Reduction
baseline environment: Test Emerald Targeted Defect R

model id: Gain_From_Early_Defect_Detection
entity: iDotCom eBusiness
baseline currency: United States Dollars

process step	*failure cost/year*		*savings*	*defects found/yr*		*fail cost / defect*	*removal eff. %*		*defects left*	
	base	*SPI*		*base*	*SPI*		*base*	*SPI*	*base*	*SPI*
Baseline Env Name: **Test Emerald Targeted Defect Removal**										
Informal design reviews	$53,550	$53,550	$0	2142	2142	$25	35.00%	35.00%	3978	3978
Formal code inspection	$129,300	$142,200	($12,900)	2586	2844	$50	65.00%	71.50%	1392	1134
New function testing	$83,600	$71,400	$12,200	418	357	$200	30.00%	31.50%	974	777
Regression testing	$67,200	$53,700	$13,500	224	179	$300	23.00%	23.00%	750	598
System testing	$135,000	$118,500	$16,500	270	237	$500	36.00%	39.60%	480	361
Customer acceptance testing	$57,600	$43,200	$14,400	72	54	$800	15.00%	15.00%	408	307
Field use	$612,000	$460,500	$151,500	408	307	$1,500	100.00%	100.00%	0	0
Summary for 'Baseline Env' = Test Emerald Targeted Defect Removal (7 detail records)										
Sum	*$1,138,250*	*$943,050*	*$195,200*	*6120*	*6120*					
Grand Total	*$1,138,250*	*$943,050*	*$195,200*	*6120*	*6120*					

Cost-Benefit Analysis Report

CBA title: **Test CBA of Emerald Targeted Defect Reduction**

Entity: iDotCom eBusiness **cba no** 35

CBA description: Determine the cost-benefits of using Emerald for Targeted Defect Reduction

Discount rate 9.00% **Time horizon in years** 5 **Currency** United States Dollars

Type of decision supporte Whether or not to implement a single SPI

How analysis will be used Cost justify funding for an initiative

Objectives to be achieved

Project start date 1/1/01 **Fiscal year start date** 1/1/01

CBA initiated date 11/1/00 **CBA completed date**

Alternative # 1 Emerald Targeted Defect Reduction

Effect	Year 1	Year 2	Year 3	Year 4	Year 5	Total
Costs						
Quality Effect->Internal Failure						
More defects found during inspections	($12,900)	($11,835)	($10,858)	($9,961)	($9,139)	($54,692)
Total Quality Effect->Internal Failure	($12,900)	($11,835)	($10,858)	($9,961)	($9,139)	($54,692)
Implementation->Tools and information systems						
Cost to acquire product or tool	($150,000)					($150,000)
Tool / system maintenance costs		($24,771)	($22,725)	($20,849)	($19,127)	($87,472)
Total Implementation->Tools and information systems	($150,000)	($24,771)	($22,725)	($20,849)	($19,127)	($237,472)
Implementation->Training						
Personnel time in training	($1,440)	($1,321)	($1,212)	($1,112)	($1,020)	($6,105)
Total Implementation->Training	($1,440)	($1,321)	($1,212)	($1,112)	($1,020)	($6,105)

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

CBA title:	Test CBA of Emerald Targeted Defect Reduction					
Effect	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Total**
Implementation->Use and operations						
Operations support	($12,000)	($11,009)	($10,100)	($9,266)	($8,501)	($50,877)
Data analysis, summarization and reporting	($12,000)	($11,009)	($10,100)	($9,266)	($8,501)	($50,877)
Total Implementation->Use and operations	($24,000)	($22,018)	($20,200)	($18,532)	($17,002)	($101,753)
Sum of Costs	($188,340)	($59,945)	($54,995)	($50,454)	($46,289)	($400,023)
Benefits						
Quality Effect->External Failure Costs						
Fewer defects found from customer use	$151,500	$138,991	$127,515	$116,986	$107,326	$642,318
Total Quality Effect->External Failure Costs	$151,500	$138,991	$127,515	$116,986	$107,326	$642,318
Quality Effect->Quality Appraisal						
Code Inspection Time Savings	$26,174	$24,013	$22,030	$20,211	$18,542	$110,970
Total Quality Effect->Quality Appraisal	$26,174	$24,013	$22,030	$20,211	$18,542	$110,970
Quality Effect->Internal Failure						
Fewer defects found during new function testing	$12,200	$11,193	$10,268	$9,421	$8,643	$51,725
Fewer defects found during regression testing	$13,500	$12,385	$11,363	$10,424	$9,564	$57,236
Fewer defects found during system testing	$16,500	$15,138	$13,888	$12,741	$11,689	$69,955
Fewer defects found during Customer Acceptance	$14,400	$13,211	$12,120	$11,119	$10,201	$61,052
Total Quality Effect->Internal Failure	$56,600	$51,927	$47,639	$43,706	$40,097	$239,968
Customer / Market Impact->Revenue Impact						
Customer satisfaction	$180,000	$165,138	$151,502	$138,993	$127,517	$763,150
Cycle Time Reduction	$102,756	$94,271	$86,487	$79,346	$72,795	$435,656

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

CBA title:	Test CBA of Emerald Targeted Defect Reduction					
Effect	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Total**
Total Customer / Market Impact->Revenue Impact	$282,756	$259,409	$237,990	$218,339	$200,311	$1,198,805
Sum of Benefits	$517,030	$474,339	$435,174	$399,242	$366,277	$2,192,061
Total Net Present Val	$328,690	$414,394	$380,178	$348,787	$319,988	**$1,792,038**
Return on Investment	2.75	7.91	7.91	7.91	7.91	**5.48**

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

CBA title: **Test CBA of Emerald Targeted Defect Reduction**

Intangible / Semitangible Effects

Effect	Comment
SEMITANGIBLE BENEFITS	
Analyze failures and help identify areas where process improvement is needed	
Clone detection and Reuse evaluation	
Enhanced understanding of field problems	
Evaluate outsourced 3rd party software	Third party software quality increases reducing the cost of usage.
Evaluate the quality of software products	e.g., design specifications, software, test plans, user documentation
Focused reviews and reengineering on modules more likely to have faults	
Improve decision making	
Improve managing process change	Provides objective data to quantify and support change decisions.
Improved patch design decisions	Incorporate the risk of defective patches
Improved productivity, reduced rework, detecting low productivity	
Improved project planning and estimating	
Improved resource allocation decisions and staffing assignments	
Keeping project on schedule	Schedule slippage leads to inefficient use of resources and increases overall cost of the project. Metrics inform management decisions early and often to enable corrective actions
Promote SEI Capability Maturity Model (CMM) progression	
Reduce risk of financial penalties	In some environments, missing schedules doesn't just lead to inefficient use of resources, but also to missed award fees or to direct assessment of penalties.
Targeted more effective testing efforts and test automation efforts	
Tracking development status in terms of complexity	

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

CBA title: **Test CBA of Emerald Targeted Defect Reduction**

Validate the results of process improvement initiatives

INTANGIBLE BENEFITS

Ability to self regulate, self audit

Avoid cost-overruns

Avoid losing market share

Bring unknown factors to the designer regarding module usage.

More effective design decisions can be made regarding product usage.

Enterprise distribution of information on which to make decisions

Showcasing the Emerald toolset and reselling, best in class

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

CBA title: **Test CBA of Emerald Targeted Defect Reduction**

CBA Parameters for SPI Alternative # 1 ***SPI:*** Emerald Targeted Defect Reduction

SPI Cost Parameter

Annual value of cycle reduction	cycle_reduct_value	=proj_days_saved_yr * cycle_day_value
How many administrators needed for the tool?	num_tool_administrators	1
Number of tool users	num_tool_users	5
Number of people to be trained	num_trainees	=num_tool_users + num_tool_administrators
Pre-release labor cost savings	prerelease_savings	=(baseline_insp_cost - SPI_insp_cost) +SPI_InternalFailureSavings
Portion of the reviewed code estimated to be of high risk ("red")	red_ratio	0.2
Tool administration hours per week	tool_admin_hrs_per_wk	4
Cost of client license for SPI tool	tool_client_license_cost	$4,000
Initial cost of tools to facilitate SPI	tool_cost	=tool_client_license_cost*num_tool_users+tool_server_license_cost
Tool maintenance cost	tool_maint_cost	=tool_cost*tool_maint_pct
Annual tool maintenance percentage	tool_maint_pct	18%
Server license cost	tool_server_license_cost	$130,000
Training hours per trainee	training_hrs_per_trainee	4
The annual cost of using the tool	use_cost_yr	=use_hrs_wk * loaded_labor_rate * weeks_per_year
Effort hours spent reviewing metrics information each week	use_hrs_wk	4
Weeks per year	weeks_per_year	50

SPI Customer / Market Parameters

Value of each cycle day saved	cycle_day_value	$9,000
Project days saved per year	proj_days_saved_yr	= staff_days_saved_yr / staff_day_to_proj_day
Additional business per year under SPI scenario	SPI_add_business_yr	$80,000
Annual repeat business under SPI scenario	SPI_repeat_business_yr	$1,100,000

SPI Productivity Parameter

Average effort to inspect a low risk ("green") module	grn_effort	2
Average lines of code covered per inspection of a low risk module	grn_loc_insp	400
Average effort hours to inspect a high risk ("red") module	red_effort	4
Average lines of code reviewed per inspection of a high risk ("red") module	red_loc_insp	150
Staff days saved per year	staff_days_saved_yr	=staff_hrs_saved_yr / staff_proj_hrs_day

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

CBA title: **Test CBA of Emerald Targeted Defect Reduction**

Staff hours saved per year	staff_hrs_saved_yr	=prerelease_savings / loaded_labor_rate

SPI Quality Assessment Parameter

Internal failure cost savings	SPI_InternalFailureSavings	$43,700

Baseline Environments

SPI: Emerald Targeted Defect Reduction

Baseline Env	*Organization*
Test Emerald Targeted Defect Removal	iDotCom eBusines

CMM level/sublvl 2 Repeatable 0 % *baseline type* Consolidate

subindustry Systems *currency unit* United States Dollars

Baseline Environment Parameters

Code Parameters

Average effort hours per inspection	avg_effort_insp	3.5
Average lines of code per baseline inspection	avg_loc_insp	250
Baseline percent of code reviewed each year	base_pct_of_code_reviewe	80%
Churned lines of code	churned_loc	8350
Percentage of code "in play"?	code_churn_pct	14.27%
Current code base size in Lines of Code	current_size_LOC	58500
New lines of code	new_loc	108500
New or changed lines of code	New_or_changed_LOC	116850
Number of projects developed within one year	num_projects	2
Reviewed lines of code	reviewed_loc	=New_or_changed_LOC * base_pct_of_code_reviewed
Emerald inspection cost	SPI_insp_cost	=reviewed_loc * loaded_labor_rate * (red_ratio * (red_effort / red_loc_insp) + (1 - red_ratio) * (grn_effort / grn_loc_insp))

Cost Parameters

Baseline inspection cost	baseline_insp_cost	=reviewed_loc * (avg_effort_insp / avg_loc_insp) * loaded_labor_rate
Loaded hourly labor rate	loaded_labor_rate	$60
Staff day to project day	staff_day_to_proj_day	= dev_staff_size / num_projects
Staff hours per day on project	staff_proj_hrs_day	6

Customer / Market Parameters

Baseline repeat business per year	baseline_repeat_business_yr	$1,000,000

General Development Parameters

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

CBA title:	Test CBA of Emerald Targeted Defect Reduction	
Size of development staff	dev_staff_size	34

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

F.2 Cleanroom Example for Statistical Testing

Cost-Benefit Analysis Report

CBA title: Test CBA for statistical testing

Entity: iDotCom eBusiness **cba no** 32

CBA description: An example cost-benefit analysis to estimate the value of statistical testing.

Discount rate 9.00% **Time horizon in years** 5 **Currency** United States Dollars

Type of decision supporte Whether or not to implement a single SPI

How analysis will be used Cost justify funding for an initiative

Objectives to be achieved Improve release decisions

Project start date 11/1/00 **Fiscal year start date** 1/1/01

CBA initiated date 11/1/00 **CBA completed date**

Alternative # 1 Statistical Testing

Effect	Year 1	Year 2	Year 3	Year 4	Year 5	Total
Costs						
Implementation->Tools and information systems->Cost to acquire product or tool						
Purchase of Usage Modeling tool	($35,000)	($32,110)	($29,459)	($27,026)	($24,795)	($148,390)
Total Implementation->Tools and information systems->Cost t acquire product or tool	($35,000)	($32,110)	($29,459)	($27,026)	($24,795)	($148,390)
Implementation->Training->Costs to purchase training						
Training class	($7,500)					($7,500)
Cost of consultants and coaching	($1,600)					($1,600)
Total Implementation->Training->Costs to purchase training	($9,100)					($9,100)

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

CBA title:	Test CBA for statistical testing					
Effect	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Total**
Quality Effect->Quality Appraisal->Testing						
Time to analyze, develop and maintain usage model	($314,100)	($288,165)	($264,372)	($242,543)	($222,516)	($1,331,696)
Time to interface usage test generator to testing tool	($4,800)					($4,800)
Total Quality Effect->Quality Appraisal->Testing	($318,900)	($288,165)	($264,372)	($242,543)	($222,516)	($1,336,496)
Implementation->Tools and information systems->Tool / system maintenance costs						
Annual maintenance fees for Usage Modeling tool	($5,250)	($4,817)	($4,419)	($4,054)	($3,719)	($22,259)
Total Implementation->Tools and information systems->Tool / system maintenance costs	($5,250)	($4,817)	($4,419)	($4,054)	($3,719)	($22,259)
Implementation->Training						
Personnel time in training	($12,000)					($12,000)
Total Implementation->Training	($12,000)					($12,000)
Sum of Costs	($380,250)	($325,092)	($298,249)	($273,623)	($251,030)	($1,528,245)
Benefits						
Production Cost Impact->Management->Decision Support						
Improved support for release decisions	$155,000	$142,202	$130,460	$119,688	$109,806	$657,157
Total Production Cost Impact->Management->Decision Support	$155,000	$142,202	$130,460	$119,688	$109,806	$657,157
Production Cost Impact->Maintenance						
Reduced maintenance cost	$58,320	$53,505	$49,087	$45,034	$41,315	$247,260

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

CBA title:	Test CBA for statistical testing					
Effect	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**	**Total**
Total Production Cost Impact->Maintenance	$58,320	$53,505	$49,087	$45,034	$41,315	$247,260
Production Cost Impact->Documentation->Requirements						
Reduced requirements creep cost	$57,081	$52,368	$48,044	$44,077	$40,437	$242,006
Total Production Cost Impact->Documentation->Requirements	$57,081	$52,368	$48,044	$44,077	$40,437	$242,006
Customer / Market Impact->Revenue Impact						
Customer retention	$20,000	$18,349	$16,834	$15,444	$14,169	$84,794
Reduced cycle time value	$15,000	$13,761	$12,625	$11,583	$10,626	$63,596
Total Customer / Market Impact->Revenue Impact	$35,000	$32,110	$29,459	$27,026	$24,795	$148,390
Quality Effect->Quality Appraisal->Testing						
Automated test case generation	$195,000	$178,899	$164,128	$150,576	$138,143	$826,745
Effective, efficient testing	$32,076	$29,428	$26,998	$24,769	$22,723	$135,993
Total Quality Effect->Quality Appraisal->Testing	$227,076	$208,327	$191,125	$175,344	$160,866	$962,739
Sum of Benefits	$532,477	$488,511	$448,175	$411,170	$377,220	$2,257,552
Total Net Present Val	$152,227	$163,419	$149,926	$137,546	$126,189	**$729,307**
Return on Investment	1.40	1.50	1.50	1.50	1.50	**1.48**

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

CBA title: **Test CBA for statistical testing**

CBA Parameters for SPI Alternative # 1 *SPI:* Statistical Testing

SPI Cost Parameter

Consulting direct contribution percent	consult_direct_pct	90%
Consultant hourly rate	consult_hr_rate	$100
Consulting hours to implement SPI	consulting_hrs	160
Effort hours to adapt tool to particular environment	interface_tool_hrs	80
Maintenance cost savings	maint_cost_savings	=maint_cost_yr * maint_reduct_pct
Number of tool users	num_tool_users	=num_usage_modelers
Number of usage modelers	num_usage_modelers	5
Savings from reduction of requirements creep	reduce_rqmts_creep_savings	=software_budget_per_yr * 12 * (design_phase_budget_pct * design_rel_cost_fp * design_phase_creep_mth * design_creep_reduce + code_phase_budget_pct * code_rel_cost_fp * code_phase_creep_mth * code_creep_reduce + test_phase_budget_pct * test_rel_cost_fp * test_phase_creep_mth * test_creep_reduce)
Cost of client license for SPI tool	tool_client_license_cost	$5,000
Initial cost of tools to facilitate SPI	tool_cost	=tool_client_license_cost * num_tool_users + tool_server_license_cost
Tool maintenance cost	tool_maint_cost	=tool_cost*tool_maint_pct
Annual tool maintenance percentage	tool_maint_pct	15%
Server license cost	tool_server_license_cost	$10,000
Training cost per modeler	training_cost_per_modeler	$1,500
Training hours per trainee	training_hrs_per_trainee	40

SPI Customer / Market Parameters

Value of each cycle day saved	cycle_day_value	1000
Project days saved per year	proj_days_saved_yr	15
Additional business per year under SPI scenario	SPI_add_business_yr	$10,000
Annual repeat business under SPI scenario	SPI_repeat_business_yr	$20,000

SPI Decision parameters

Percentage of time correct decision will be made under SPI scenario	SPI_correct_prediction	95%

CBA title: **Test CBA for statistical testing**

SPI cost per decision	SPI_cost_per_decision	=payoff_decision1_state1 * SPI_decision1_state1_prob + payoff_decision1_state2 * SPI_decision1_state2_prob + payoff_decision2_state1 * SPI_decision2_state1_prob + payoff_decision2_state2 * SPI_decision2_state2_prob
Annual SPI decision savings	SPI_decision_savings_yr	=(SPI_cost_per_decision - baseline_cost_per_decision) * decisions_per_year
SPI probability of decision 1 and state 1	SPI_decision1_state1_prob	=state_1_prob * SPI_correct_prediction
SPI probability of decision 1 and state 2	SPI_decision1_state2_prob	=(1 - state_1_prob) * (1 - SPI_correct_prediction)
SPI probability of decision 2 and state 1	SPI_decision2_state1_prob	=state_1_prob * (1 - SPI_correct_prediction)
SPI probability of decision 2 and state 2	SPI_decision2_state2_prob	=(1 - state_1_prob) * SPI_correct_prediction

SPI Productivity Parameter

Automatic test generation savings 1	auto_test_gen_savings1	=avg_test_case_prep_hrs * loaded_labor_rate *SPI_num_test_cases_yr
Coding phase creep reduction	code_creep_reduce	10%
Percent of requirements creep reduced in design phase	design_creep_reduce	10%
Percent reduction in maintenance cost	maint_reduct_pct	10%
How many KLOCs are subject to the SPI each year	SPI_KLOC_yr	=new_loc/1000
Number of test cases applicable to the SPI each year	SPI_num_test_cases_yr	6,500
Test phase creep reduction	test_creep_reduce	10%
Percentage of testing budget that would be displaced by SPI	test_gen_pct	5%
Annual effort hours to develop usage models	usage_model_dev_hrs	=36 * KLOCS_per_year + 15

SPI Quality Assessment Parameter

Percent improvement in testing productivity	test_improve_pct	10%

Baseline Environments **SPI:** Statistical Testing

Baseline Env	*Organization*
Baseline for Statistical Testing template	iDotCom eBusines

CMM level/sublvl 3 Defined 0 % *baseline typ* Consolidate

subindustry Systems *currency unit* United States Dollars

Baseline Environment Parameters

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

CBA title: **Test CBA for statistical testing**

Code Parameters

Churned lines of code	churned_loc	45,000
Percentage of code "in play"?	code_churn_pct	15.00%
Percent of new requirements per month of coding phase	code_phase_creep_mth	1.25%
Current code base size in Lines of Code	current_size_LOC	300,000
Percent of requirements introduced during design phase per month	design_phase_creep_mth	3%
New or changed KLOCS per year	KLOCS_per_year	=New_or_changed_LOC / 1000
New lines of code	new_loc	100,000
New or changed lines of code	New_or_changed_LOC	145,000
Percent of new requirements per month of testing phase	test_phase_creep_mth	0.5%

Cost Parameters

Average time to prepare a test case in hours	avg_test_case_prep_hrs	0.5
Relative cost per function point for requirements introduced during coding	code_rel_cost_fp	1.5
Relative cost per function point for requirements introduced during design	design_rel_cost_fp	1.25
Loaded hourly labor rate	loaded_labor_rate	$60
Maintenance cost per year	maint_cost_yr	=maint_effort_pct * software_budget_per_yr
Software development budget per year	software_budget_per_yr	$2,916,000
Testing cost per year	test_cost_yr	=test_effort_pct * software_budget_per_yr
Percent of budget spent on testing	test_effort_pct	11%
Relative cost per function point for requirements introduced during testing	test_rel_cost_fp	2.5

Customer / Market Parameters

Baseline repeat business per year	baseline_repeat_business_yr	$10,000

Decision parameters

Probability that code is high quality at time of release decision	code_high_quality_prob	50%
Payoff for releasing good quality code on-time	payoff_release_good_quality	$200,000
Payoff for releasing poor quality code	payoff_release_poor_quality	-$420,000
Payoff for delaying release of good quality code	payoff_test_good_quality	-$300,000
Payoff for delaying the release of poor quality code for continued testing	payoff_test_poor_quality	-$300,000

Effort distribution

Percent of budget spent during coding phase	code_phase_budget_pct	21%
Percent of budget spent during design phase	design_phase_budget_pct	23%

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

CBA title: **Test CBA for statistical testing**

Percent of budget spent on software maintenance	maint_effort_pct	20%
Percentage of budget spent during testing phase	test_phase_budget_pct	30%

SPI Decision parameters

Baseline probability of decision 1 and state 1	base_decision1_state1_prob	=state_1_prob *baseline_correct_prediction
Baseline probability of decision 1 and state 2	base_decision1_state2_prob	=(1 - state_1_prob) * (1 - baseline_correct_prediction)
Baseline probability of decision 2 and state 1	base_decision2_state1_prob	= state_1_prob * (1 - baseline_correct_prediction)
Baseline probability of decision 2 and state 2	base_decision2_state2_prob	=(1 - state_1_prob) * baseline_correct_prediction
Percentage of time correct decision made under baseline scenario	baseline_correct_prediction	85%
Baseline cost per decision	baseline_cost_per_decision	=payoff_decision1_state1 * base_decision1_state1_prob + payoff_decision1_state2 * base_decision1_state2_prob + payoff_decision2_state1 * base_decision2_state1_prob + payoff_decision2_state2 * base_decision2_state2_prob
Decisions made per year	decisions_per_year	5
Payoff for decision 1 when state 1 occurs	payoff_decision1_state1	=payoff_release_good_quality
Payoff for decision 1 when state 2 occurs	payoff_decision1_state2	=payoff_release_poor_quality
Payoff for decision 2 when state 1 occurs	payoff_decision2_state1	=payoff_test_good_quality
Payoff for decision 2 when state 2 occurs	payoff_decision2_state2	=payoff_test_poor_quality
State 1 probability	state_1_prob	=code_high_quality_prob

Disclaimer: The end user is responsible for verifying the reasonableness of the cost and benefit estimates and ensuring that benefits are not double counted or overstated.

Appendix G

Acronyms

ASQC	American Society for Quality Control
CBA	Cost-Benefit Analysis
CME	Certainty Monetary Equivalent
CMM	Capability Maturity Model
COCOMO	COnstructive COst MOdel
COSQ	Cost of Software Quality metric
EMV	Expected Monetary Value
FP	Function Point
GQM	Goal/Question/Metric Paradigm
IRR	Internal Rate of Return
ISO	International Standards Organization
KLOC	Kilo-Lines of Code
KPA	Key Process Area
KSLOC	Kilo-Source Lines of Code
MIS	Management Information Systems
MPIS	Most Productive Increment Size

MPP	Modern Programming Practices
MPSS	Most Productive Scale Size
MTTF	Mean Time to Fail
NPV	Net Present Value
PM	Person-months of effort
QA	Quality Assurance or Quality Assessment depending on the context.
QIP	Quality Improvement Paradigm
ROI	Return on Investment
ROSQ	Return on Software Quality metric.
SA-CMM	Software Acquisition Capability Maturity Model
SBS	Sequence-based specification
SD	System dynamics
SEI	Software Engineering Institute
SPI	Software Process Improvement
TED	Task Element Decomposition

Appendix H

Glossary

Bayesian decision analysis A structured approach to evaluating choices with an uncertain pay-off for those choices. Decision analysis is useful for evaluating the value of increased information on choices involved in routine decision making.

capability maturity model (CMM) A software process maturity model developed by the Software Engineering Institute that describes the stages through which software organizations evolve as they improve their software processes. The model consists of five maturity levels and serves as a guide to help organizations select process improvement strategies.

certainty monetary equivalent (CME) The average price members of society would be willing to pay for a potential cost or benefit that has a degree of uncertainty as to whether the cost or benefit will be realized.

Cleanroom software engineering A collection of principles and processes aimed at the economical production of high quality software. Cleanroom processes include sequence-based functional specification, functional verification, incremental development, and statistical usage testing.

cost-benefit analysis (CBA) An evaluation of the net benefits associated with one or more proposed alternatives for achieving a defined goal.

defect potential The total number of defects per function point that might be expected to occur in a software application.

defect removal efficiency The percentage of defects that will be removed by a quality appraisal step. This percentage is calculated as the number of defects found and repaired by the step divided by the total number found in the software through the first year of use.

discount rate A rate which, when applied to future costs and benefits, yields the present value of those costs or benefits.

function point (FP) A metric for measuring the size of a software application by measuring visible aspects of the application's functionality. On average, it takes about 128 lines of C code to develop a function point.

functional verification A systematic approach to team software verification where the correctness of a software product is verified against its specification using correctness conditions and reasoning based on function theory.

incremental development The organization of a large software project into a series of smaller, cumulative, and more manageable increments.

internal rate of return (IRR) The rate used to discount the future which would make the NPV of the project equal to zero. A proposal with an IRR that exceeds a predetermined social discount rate (e.g., cost of capital) is deemed acceptable.

key process area A set of related activities that are considered important for achieving a process capability as defined by the SEI's Capability Maturity Model.

net present value (NPV) A method for discounting projected costs or benefits which will occur in the future. Essentially, the NPV recognizes that money has a time value (even in the absence of inflation). The formula for NPV is

$$NPV = \sum_{t=0}^{n} \frac{B_t - C_t}{(1+r)^t}: \qquad \text{(EQ 60)}$$

where

B_t is the dollar value of benefits received at time t,

C_t the costs incurred at time t,

r the discount rate,

n the life of the project, and

t is time in units such as years or months.

quality appraisal step An activity to identify potential defects with a software product. Examples of quality appraisal steps include: unit testing, inspection, functional verification, beta testing, acceptance testing, and system testing.

return on investment (ROI) The Return on Investment (ROI) (also called the Benefit-Cost Ratio (B/C) or a profitability index) is the ratio of discounted benefits to discounted costs. The formula for computing the ROI (or B/C) is

$$ROI = B/C = \frac{\sum_{t=0}^{n} \frac{B_t}{(1+r)^t}}{\sum_{t=0}^{n} \frac{C_t}{(1+r)^t}}$$

In the software engineering literature, the ROI is often expressed without discounting future values (i.e., $r = 0$).

sequence-based specification A systematic sequence enumeration process for developing complete, consistent, and traceably correct software specifications.

Software Engineering Institute (SEI) A federally funded research institute of Carnegie Mellon University that was established by the Department of Defense to help facilitate transfer of software engineering technology.

software process The Software Engineering Institute defines *software process* as "as set of activities, methods, practices, and transformations to develop and maintain software and the associated products, (e.g., project plans, design documents, code, test cases, and user manuals.)"

software process simulation model An abstract representation of an actual software process that can be simulated computationally.

statistical usage testing An approach to testing of software that views software testing as a statistical problem that requires sampling. A usage model is constructed to characterize how the system will be used, and is represented as a discrete time Markov chain. Randomly generated test cases from the usage model are used to evaluate the software under test.

Vita

Daniel T. Fetzer is from Elizabethton, a small town nestled in the mountains of north-east Tennessee. He graduated from Elizabethton High School where he played trombone in the band and keyboards in the stage band. He attended East Tennessee State University (ETSU) with a major in Music. After a three year break to pursue a career as a professional musician, he resumed his education at ETSU and earned a Bachelor of Science in Computer Science.

Upon graduation, he accepted a software analyst / developer position with Oak Ridge Associated Universities where he worked for over nine years. He subsequently worked as a Computing Specialist with Lockheed Martin Energy Systems in Oak Ridge for eight years. While working in Oak Ridge, he enrolled in the evening school at the University of Tennessee (UT) to pursue a Master's degree in Computer Science which he was awarded in May 1992. A few years later he re-enrolled in the University of Tennessee to pursue a Doctorate of Philosophy in Computer Science which he was awarded in December 2000.

Presently, he is working for the Reliametrics software reliability organization of Nortel Networks in the Research Triangle region of North Carolina.