Evaluating the Cost of Software Quality

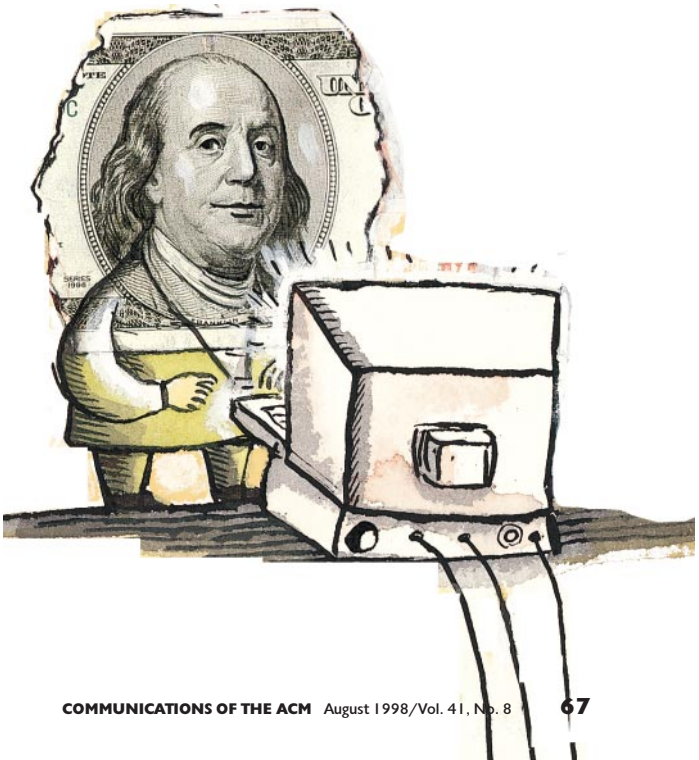
The time has come to financially justify investments in software quality improvements, just like we justify other software projects.

Sandra A. Slaughter, Donald E. Harter, and Mayuram S. Krishnan

The explosive growth of the software industry in recent years has focused attention on the problems long associated with software development: uncontrollable costs, missed schedules, and unpredictable quality. To remain competitive, software firms must deliver high quality products on time and within budget. However, to bring their products more quickly to market, software managers may avoid quality improvement processes such as design reviews and code inspections, believing that these processes only add time to the development cycle. Certainly the economics of improving quality are not well understood in the software development world [9].

There is some confusion about the business value of quality even outside the software development context. On the one hand, there are those who believe that it is economical to maximize quality. This is the "quality is free" perspective espoused by Crosby [7], Juran and Gryna [8], and others. Their key argument is that as the voluntary costs of defect prevention are increased, the involuntary costs of rework decrease by much more than the increase in prevention costs. The net result is lower total costs, and thus quality is free. On the other hand, there are those who believe it is uneconomical to have high levels of quality and assume they must sacrifice quality to achieve other objectives such as reduced development cycles. For example, a study of adoption of the Software Engineering Institute's Capability Maturity Model (CMM™) reports the following quote from a software manager: "I'd rather have it wrong than have it late. We can always fix it later" [11].

Experiences in manufacturing relating to the cost



Important questions arise concerning whether and how much to invest in specific software quality improvement initiatives.

and return of quality improvements suggest that there are diminishing returns to quality expenditures [12]. However, quality improvements can often result in quantifiable cost savings that outweigh the money spent on the quality efforts. A key management problem therefore is how to make profitable decisions on quality expenditures. This problem is particularly salient in software development due to limited empirical evidence on the economics of software quality.

In this article, we focus on evaluating the cost of quality and return on quality from the perspective of software development. We introduce three new metrics in the software engineering economics context: cost of software quality (COSQ), return on software quality (ROSQ), and software quality profitability index (SQPI).[1] We then report the results from a detailed longitudinal study on the economics of software quality at BDM International, a major information technology company.[2] Our analysis yields a number of important insights for software managers who are interested in improving their decisions on software quality expenditures.

Cost of Software Quality

The costs of quality as originally articulated by Juran and Gryna are those that would be eliminated if all workers were perfect in their jobs. Quality costs are important because every dollar and labor hour not spent on rework can be used for making better products more quickly or for improving existing products and processes. The costs of quality are divided into two major types: *conformance* and *nonconformance*.

The cost of conformance is the amount spent to achieve quality products. It is further divided into costs of prevention and appraisal. *Prevention costs* are those associated with preventing defects before they happen. In software development, examples of prevention costs include the costs of training staff in design methodologies, quality improvement meetings, and software design reviews. *Appraisal costs* include measuring, evaluating, or auditing products to assure conformance to quality standards and performance. For software, examples of appraisal costs include code inspections, testing, and software measurement activities.

The cost of nonconformance includes all expenses that are incurred when things go wrong. *Internal failure* costs occur before the product is shipped to the customer. For software these include the costs of rework in programming, reinspection, and retesting. *External failure* costs arise from product failure at the customer site. For software, examples include field service and support, maintenance, liability damages, and litigation expenses.

So how can companies reduce the costs of software quality? A basic strategy is to drive failure costs to zero, invest in the "right" prevention activities to bring about improvement, reduce appraisal efforts as quality improves, and continue to evaluate and alter preventive efforts for further improvement [5]. The idea behind this approach is that real software failure costs can be measured and then reduced through the proper analysis of cause and effect. Elimination of root causes means identifying and permanently fixing defects as early in the software life cycle as possible, because the cost of correction increases the later in the software process the defect is discovered and corrected. As software failure costs are reduced, appraisal costs can also be reduced, and the total software quality costs decrease.

Important questions then arise concerning whether and how much to invest in specific software quality improvement initiatives. It is useful to approach these questions from a financial return on investment (ROI) perspective. We refer to this as the return on software quality.

Return on Software Quality (ROSQ)

The rationale behind ROSQ is that software quality expenditures must be financially justified. Increasingly, the chief financial officers in many companies are promoting disciplines for financial evaluation to

[1]Our work extends an important stream of research on metrics for assessing the economics of software engineering. Readers are referred to the work of Chidamber and Kemerer on object-oriented metrics [6]; Banker, Kauffman, Wright, and Zweig on metrics for software reuse leverage and value [2]; Banker and Slaughter [3] and Banker, Chang, and Kemerer [1] on project scale size in software development and maintenance; and Mukhopadhyay and Kekre on features for software cost estimation [10].

[2]The authors would like to thank BDM International for providing access to archived data for this study.

encourage investments that yield the greatest response for limited resources. Such disciplines are particularly important in the context of software engineering, as software expenditures account for larger portions of capital budgets. Software quality is an investment that should provide a financial return relative to the initial and ongoing expenditures in the software quality improvement initiatives. One way to evaluate software quality improvement efforts is to consider them in terms of specific initiatives. Examples of software quality improvement initiatives include implementation of design reviews, testing and debugging tools, code walkthroughs, and quality audits. Initiatives require an initial investment—the software quality investment (SQI)—that includes the initial expenses for training, tools, effort, and materials required to implement the quality initiative. There are also ongoing expenditures for meetings, tool upgrades, and training that are required to maintain the quality process once it is in place. We call this software quality maintenance (SQM). Finally, each software quality improvement initiative should result in annual revenues. These software quality revenues (SQR) can be derived from the projected increases in sales or estimated cost savings due to the software quality improvement.

The return on the software quality initiative is the net present value of the software quality revenues and costs or cash flows (NPVCF) divided by the net present value of the initial investment and ongoing maintenance costs for the software quality initiative (NPVIC). More formally, by selecting a financial discounting factor (r) that reflects the company's weighted average cost of capital, we can calculate the ROSQ over T periods of time for which the project yields value for the firm. Related to the concept of financial ROI is the SQPI, which is the ratio of the present value of the difference between the software quality revenues and costs divided by the software quality investment.[3]

A value greater than 1 for the SQPI implies that the initiative will create value that exceeds its investment. SQPI provides a method for comparing the return on a number of initiatives. When there are limited funds for investment, only the highest value SQPI initiatives are selected. To illustrate the application of these concepts, we examine the COSQ and ROSQ for software quality initiatives at BDM International.

Evaluating COSQ and ROSQ at BDM

BDM International is a $1 billion per year IT company. (In December 1997 BDM International was acquired by TRW.) From 1985 to 1994, BDM's Systems Integration group in Dayton, Ohio developed approximately 3.5 million lines of code for the requirements determination portion of a material requirements planning (MRP) system. The impetus of software quality improvement at BDM for this project was its fixed price incentive contract. BDM agreed to absorb any costs above a ceiling price, but would retain a percentage of any savings below the original estimated cost that resulted from improved efficiency. BDM focused on improving quality in order to increase efficiency and minimize software costs.

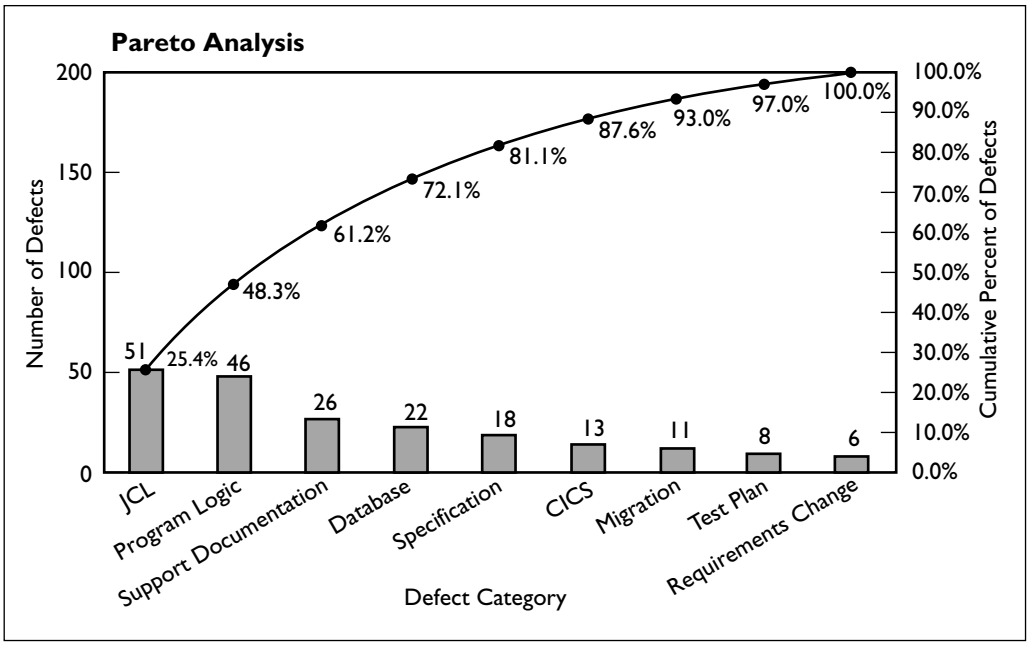


Figure 1. Identifying process improvement opportunities using Pareto analysis

Major Software Quality Initiatives. To reduce defect rates over the life of the project, four major

[3]Contact the authors for the formulas for COSQ, ROSQ, and SQPI.

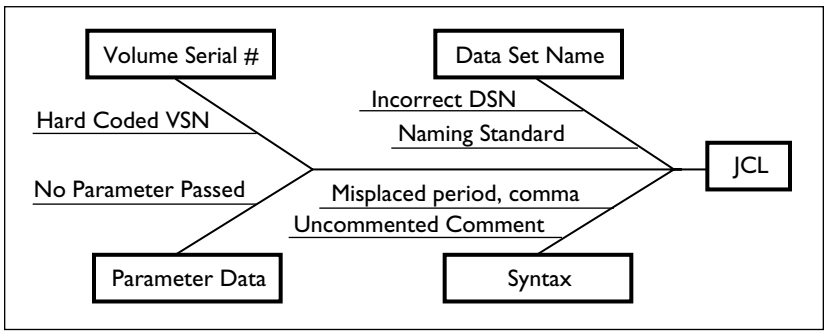


Figure 2. Cause-effect analysis using Fishbone chart

software quality improvement initiatives were implemented:

- Process Improvement #1: Creation of life-cycle development standards and the introduction of computer-aided software engineering (CASE) tools.
- Process Improvement #2: Increasing minimum educational requirements for hiring, integration of BDM's Software Blueprint® methodology with the CASE tools, creation of detailed style guides for documentation, and institutionalization of weekly program management status reviews.
- Process Improvement #3: Seamless integration of the CASE technology with the publications department, addition of schedule and performance metrics, automated development cost estimation, automated software configuration, and Pareto analysis.
- Process Improvement #4: Cycle time analysis and development of an automated support cost estimation methodology.

BDM's basic objective was to drive failure costs to zero by implementing quality initiatives that would dramatically reduce defect rates. For example, key problem areas were identified that were causing the majority of the defects. An early *Pareto analysis* (Figure 1) suggested that most of the defects were related to job control language (JCL) errors.

To analyze the cause and effect of the defects arising from JCL errors, BDM used *fishbone analysis* (Figure 2). As indicated in Figure 2, the causes of the errors were incorrect syntax, parameters, volume serial numbers, and data set names. As a result of this analysis, BDM instituted increased emphasis on JCL walkthroughs, mandatory use of automated JCL check software, and team leader approval for data set names and hard coded volume serial numbers.

Impact of Quality Initiatives. Were the quality improvement initiatives over the life of the project successful at BDM? Figure 3 plots defect density (defects per 1,000 lines of code) over the 10-year development period of the project, identifying where the major quality improvement initiatives occurred.

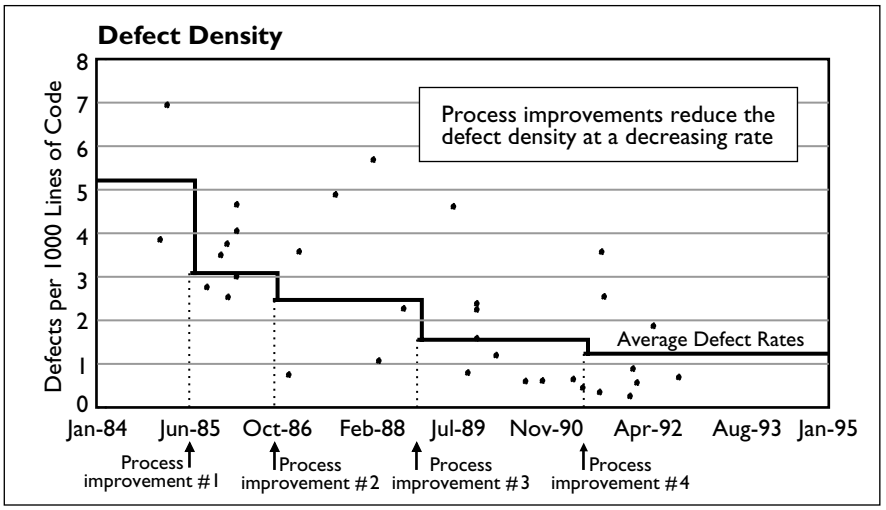


Figure 3. Defect rates vs. process improvements

As indicated in this figure, defect density improves after each quality improvement, but the data suggests that this improvement diminishes, that is, the largest reductions are toward the beginning of the project.

Total Quality Costs, Conformance, and Nonconformance Costs. We examined the behavior of total

Table 1. Return on investment analysis

Return Type	*Improvement #1*	*Improvement #2*	*Improvement #3*	*Improvement #4*
NPVCF	$1,814,370	$1,193,420	$2,099,510	$1,115,470
NPVQC	$937,550	$2,266,290	$3,580,990	$1,058,850
ROSQ	194%	53%	59%	105%
SQPI	3.83	3.65	2.96	2.74

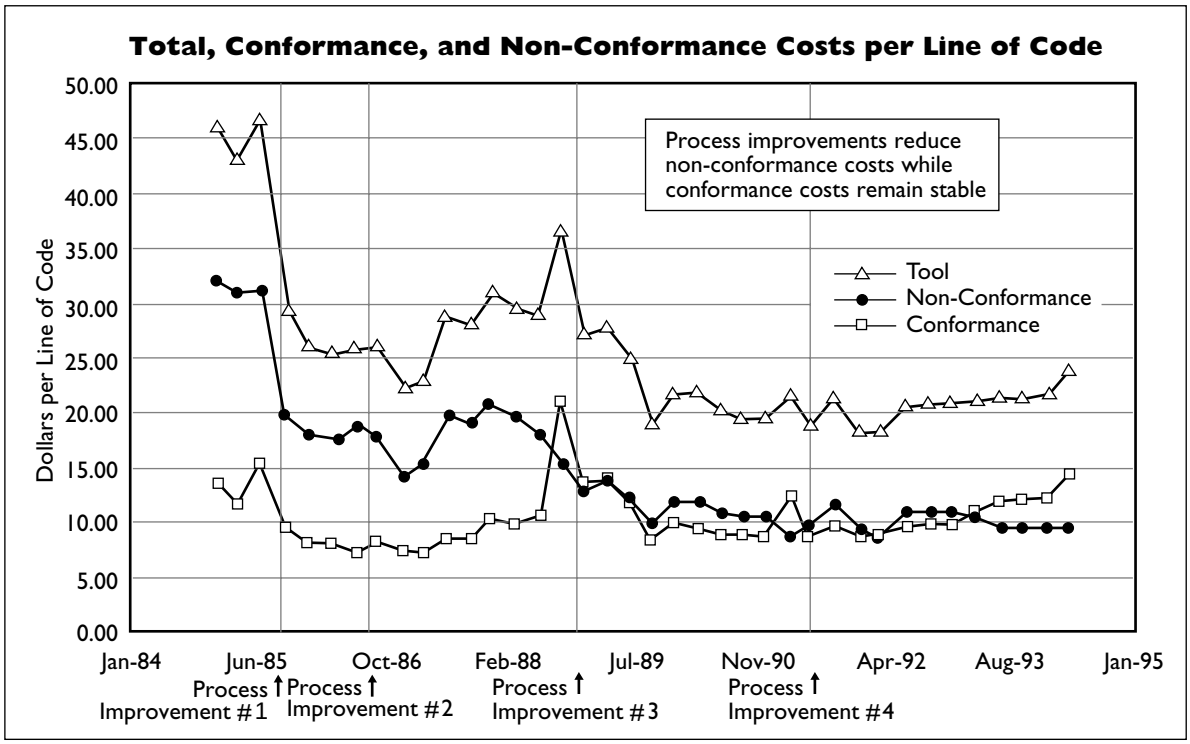


Figure 4. Costs of quality

quality costs as well as the conformance and nonconformance costs associated with this project. Total quality costs are the sum of conformance and nonconformance costs. Conformance costs include the initial and ongoing expenditures for software quality initiatives, baseline configuration management, design reviews, system-level testing, and quality assurance audits. Nonconformance costs include software debugging, configuration management migrations of software fixes, regression testing, operations support for retesting, and additional audits and reviews due to defects. Crosby [4] and others have argued that both nonconformance and conformance costs should decrease with quality improvement. As shown in Figure 4, total quality costs and nonconformance costs per line of code decrease over the life of the project (from $46 and $32 per line of code at the beginning of the project to $23 and $9 per line of code at the end). However, conformance costs appear to be largely fixed over the project (starting and ending at $14 per line of code with relatively little variance). This could reflect the limited degree to which BDM's appraisal and prevention policies were changed due in part to contractual obligations. While some minor adjustments to appraisal and prevention efforts did occur as a result of defect reduction, the costs associated with conformance did not appear to change.

Marginal Analysis of Nonconformance Costs. We then examined where the greatest nonconformance cost impacts of defect reduction occurred at BDM. Costs were tracked in 10 different software quality cost centers at BDM:

- Data Element Dictionary—for database element names, field descriptions, edit criteria
- Integration—for management of product interfaces to ensure compatibility and system-level integration
- Documentation—for system, user, and support documentation
- ADPT Support—for automated data processing technical support from hardware and system software specialists
- Operations—for computer operator support for development, testing, and production
- Quality Assurance (QA)—for auditing of processes and products
- Configuration Management (CM)—for management of baseline documents and software, and formal reviews
- Program Control—for schedule and budget tracking
- Management—for senior executive management of development, operations, and support activities
- Development—for software design, coding, and testing through customer acceptance

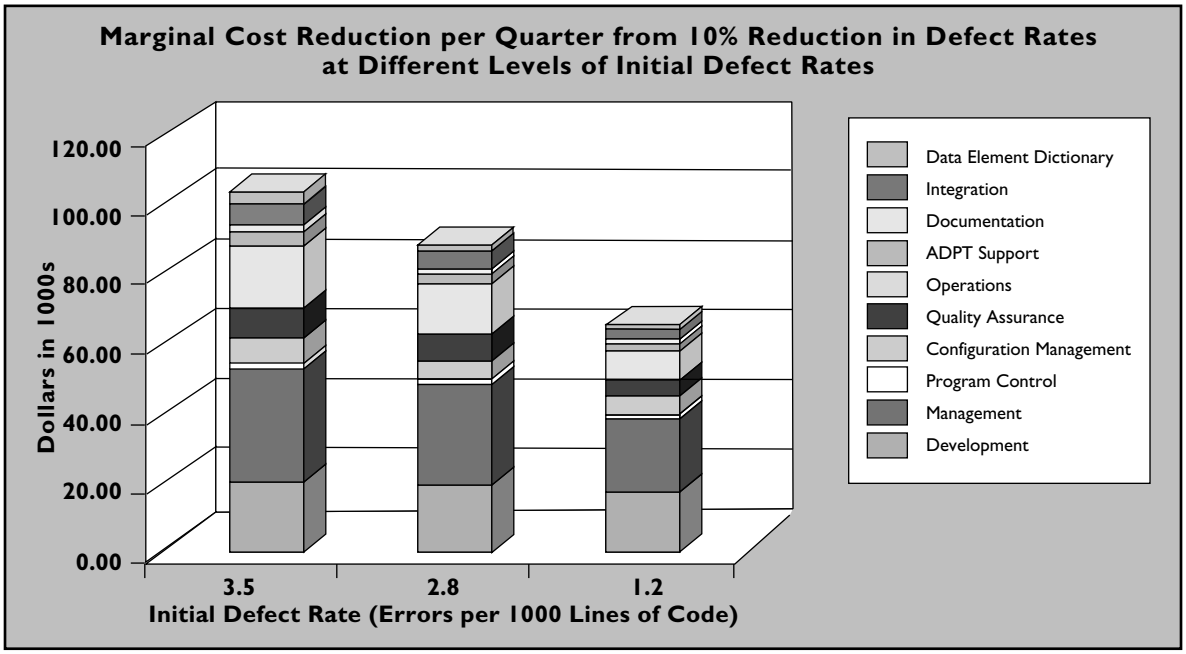


Figure 5. Marginal cost analysis

We calculated the marginal cost effects of defect reduction by estimating parsimonious log linear regressions that related each nonconformance cost type to lines of code and defects. The coefficients from these regressions were used to determine the marginal impact of 10% reductions in defect rates at different levels of initial defect rates. Marginal analysis of nonconformance costs at three different defect density levels (Figure 5) indicates that the greatest marginal cost effects of defect reduction were in the Development, Management, Operations, and Quality Assurance areas. There are a number of reasons for these findings. Development costs were directly impacted by defect reduction due to less rework in debugging, testing, and programming. Management costs experienced a large marginal impact from defect reduction due to the involvement of senior managers when errors occurred and the high cost of their time. Quality Assurance costs were also impacted because fewer defects led to less reinspection, reappraisal, and re-testing activities. Operations costs are driven by software testing, production, and maintenance support. Defect rates have a high marginal impact on operations costs because they influence allocation of operations staff support for regression testing and maintenance workload.

Return on Software Quality. To determine the return on investment of software quality improvement at BDM, we calculated the Net Present Value, ROSQ, and SQPI for the four quality initiatives (Table 1). We determined the software quality revenues in terms of nonconformance cost savings due to defect reduction. Specifically for each quality improvement, we calculated the cost savings as the difference between the nonconformance costs associated with the average defect density level prior to the investment (projecting these to the end of the project) less the nonconformance costs associated with the average improved defect level after the investment (projecting these to the end of the project). We used actual figures on defects and costs for these calculations in our analysis.

How should companies estimate the cost leverage from defect density reduction for a project a priori? One approach is to use actual historical defect and effort data from the project itself (if it is done over a long period of time) or to obtain defect and effort data from a similar project. This data can then be input into a data-driven cost estimation model (such as Capers Jones' SPQR™ or Checkpoint™) to estimate cost leverage. In BDM's case, defect density was measured before and after the first process improvement in the project to gauge the effect of the process improvement. Beginning in the third year of the project, the actual defect density and effort were compared with the estimated defect density and effort from SPQR. Actual data on process, productivity, and effort were used to update the SPQR estimation models and to recalibrate the defect density and effort predictions for the following year. Thus, BDM could estimate the cost leverage from each process improvement (after the first initiative) using data from the SPQR models.

As shown in Table 1, all four quality initiatives (numbered 1 through 4) generated a positive return as

measured by ROSQ and SQPI. However, the value for SQPI declines over time, because the investments later in the project have less time to recover their large upfront expenditures in our fixed time horizon. For both ROSQ and SQPI, the highest values are at the beginning of the project. This suggests that it would be more profitable to invest in quality early in the project so that the quality improvements could benefit more of the project.

Managerial Implications

A number of interesting observations emerged from our analysis. We found that defect density improved at BDM with each software quality initiative, but at a decreasing rate. This could reflect BDM's strategy for quality improvement, which was to focus on eliminating the major problems first. An implication of this finding is that much of the effect of quality improvement may be realized from the initial quality improvement efforts. Our analysis of BDM's software quality costs reveals that conformance costs per line of code were relatively fixed over the life of the project, which could have resulted in part from BDM's contractual obligations to keep certain processes in place. We speculate that conformance costs may be difficult to change in general, as they may involve relatively fixed components, counter to the claims of Crosby [4]. It could also be, as Campanella [5] notes, that companies must be diligent in reevaluating and adjusting their appraisal and preventive efforts so that they do not overinvest in conformance activities as quality improves. Further studies of software conformance and nonconformance costs to clarify this issue would be helpful. Finally, we found that the largest marginal returns in terms of nonconformance cost reduction at BDM were for the Development, Management, Quality Assurance, and Operations cost centers. This result suggests that it may be most cost-effective to implement software quality improvement initiatives that are specifically directed at reducing effort in these areas.

We find the larger returns from quality improvement occur early in the project (both ROSQ and SPQI are highest then) and the rest of the project can benefit from these improvements. The implication is to avoid making large investments in software quality toward the end of a project. We did not explore this at BDM, but it may be profitable to invest early in software quality improvement initiatives that have synergies with future initiatives or that make future improvements possible. For example, investing in a software metrics program at time t may enable use of other quality techniques like Pareto analysis and Statistical Process Control at time $t + 1$. Such investments create an option for future software quality improvements, and their value can be assessed using option pricing analysis [4].

The intent of our analysis has been to emphasize that software quality improvement should be viewed as an investment. It is possible to spend too much on software quality. Thus, it is important that companies financially justify each software quality improvement effort. Finally, we have seen that it is important to monitor software conformance and nonconformance costs so that conformance policies can be adjusted to reduce the *total* costs of software quality. C

References

1. Banker, R., Chang, H., and Kemerer, C. Evidence on economies of scale in software development. *Info. and Softw. Technology 36*, 5 (1994), 275–282.
2. Banker, R., Kauffman, R., Wright, C. and Zweig, D. Automating output size and reuse metrics in a repository-based computer-aided software engineering (CASE) environment. *IEEE Trans. Softw. Eng. 20*, 3 (1994), 169–187.
3. Banker, R. and Slaughter, S. A field study of scale economies in software maintenance. *Manage. Sci. 43,* 12 (Dec. 1997), 1709–1725.
4. Benaroch, M. and Kauffman, R. A case for using option pricing analysis to evaluate information technology project investments. *Info. Syst. Research,* Forthcoming.
5. Campanella, J. *Principles of Quality Costs, 2nd ed.*, ASQC Press, Milwaukee, 1990.
6. Chidamber, S. and Kemerer, C. A metrics suite for object-oriented design. *IEEE Trans. Softw. Eng. 20,* 6 (1994), 476–493.
7. Crosby, P. *Quality Is Free: The Art of Making Quality Certain*. McGraw-Hill, New York, 1979.
8. Juran, J. and Gryna, F. *Quality Control Handbook, 4th ed.*, McGraw-Hill, New York, 1988.
9. Krishnan, M.S. *Cost and Quality Considerations in Software Product Management.* Ph.D. dissertation, Graduate School of Industrial Administration, Carnegie Mellon University, 1996.
10. Mukhopadhyay, T. and Kekre, S. Software Effort Models for Early Estimation of Process Control Applications. *IEEE Trans. Softw. Eng. 18*, 10 (Oct. 1992), 915–924.
11. Paulk, M., Weber, C., Curtis, W., and Chrissis, M. *The Capability Maturity Model: Guidelines for Improving the Software Process.* Carnegie Mellon University Software Engineering Institute, 1994.
12. Wiesendanger, B. Deming's luster dims at Florida Power & Light. *J. Business Strategy 14* (Sept.–Oct. 1993), 60–61 .

Sandra A. Slaughter (sandras@andrew.cmu.edu) is an assistant professor at Carnegie Mellon University.
Donald E. Harter (harter@cmu.edu) is a Ph.D. candidate at Carnegie Mellon University.
Mayuram S. Krishnan (mskrish@umich.edu) is an assistant professor at University of Michigan Business School.