Using the Cost of Quality Approach for Software

Herb Krasner
Krasner Consulting

Cost of software quality (CoSQ) is an accounting technique that is useful to enable our understanding of the economic trade-offs involved in delivering good-quality software. Commonly used in manufacturing, its adaptation to software offers the promise of preventing poor quality but, unfortunately, has seen little use to date. This article discusses the rationale and context for using CoSQ, then defines a basic CoSQ model that differentiates the costs involved with handling nonconformances due to a lack of quality, appraisal efforts performed for the achievement of acceptable quality, and efforts to prevent poor quality from occurring.

The Nature of Software and Its Quality

Software is pervasive in modern society, but we are often unaware of its presence until problems arise. Software is one the most important and yet one of the most economically challenging technologies of the current era. As a purely intellectual product, it is among the most labor-intensive, complex, and error-prone technologies in human history. Even though many successful software products and systems exist in the world today, an overall lack of attention to quality has also led to many problematic systems that do not work right as well as to many software projects that are late, over budget, or canceled. In short, "Software Quality Matters." [1]

Although no standard industry definition exists for what constitutes good quality in software, it is generally taken to mean that a software product provides value (satisfaction) to its users, makes a profit, generates few serious complaints, and contributes in some way to the goals of humanity (or at least does no harm) [2]. Software quality is difficult to define because there is no single comprehensive and complete standard definition of its lexicon. Various aspects and terms are found in sources such as ISO 9000-3, Institute of Electrical and Electronics Engineers Software Engineering Standards, and various books on the subject. The following are the key dimensions of software quality.

- Level of satisfaction. The degree to which customers or users perceive that a software product meets their composite needs and expectations.
- Product value. The degree to which a software product has value for its various stakeholders relative to the competitive environment.
- Key attributes ("ilities"). The degree to which a software product possesses a combination of desired properties, e.g., reliability, portability, maintainability.
- Defectiveness. The degree to which a software product works incorrectly in target user environments due to debilitating operational defects.
- Process quality. In relation to the development process by which the product is produced, it means good people doing the right things in an effective way.

A definition fashioned from the above aspects should be created for your organization and for each project. Every application or business domain faces a specific set of software quality issues, and software quality must be defined accordingly. For example, mission-critical applications have extremely stringent operational needs, whereas typical information system applications must focus on general measures of customer satisfaction. It also is important for each software development project to define its specific meaning of software quality during the planning phase. Such a definition contributes to the basis for setting objectives and practical measures of quality progress and determination of readiness for release to customers. An example of such a definition is shown in Figure 1 as a Figure of Merit (FOM) Quality Factors Model.

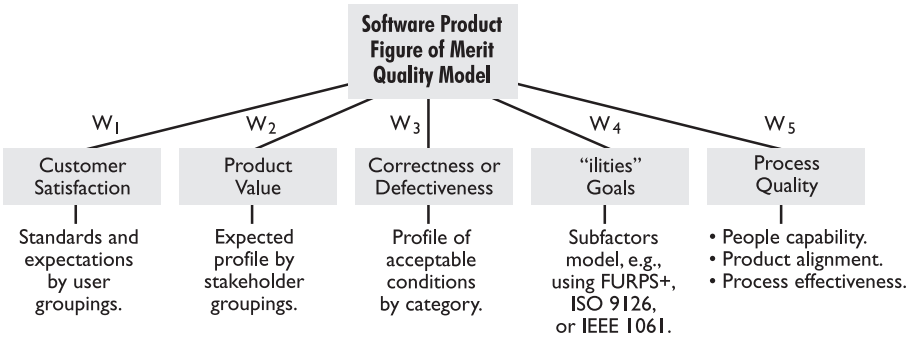
The categories and subcategories of the FOM model can be weighted as needed for use by domain or system. The FOM would be the sum of the weighted factors.

Now that software quality has been defined and its supreme importance established, it is necessary to examine the cost perspective of the *economics of software quality*, a subject in serious need of an underlying theory.

Why Is CoSQ Important Now?

If it is an organizational goal to improve business success through software quality, it is important to address answers to the following questions, which are too often not asked in today's software development groups.

Figure 1. *Software quality FOM model example.*



Major Category	Subcategory	Definition	Typical Cost Subitems
Dealing with nonconformances.	Internal nonconformances.	Quality problems detected prior to product shipment.	Pre-release defect management, rework, re-reviews, and retesting.
	External nonconformances.	Quality problems detected after product delivery.	Post-release technical support, complaint investigation, defect notification, remedial upgrades, and fixes.
Appraising the level of quality.	Discovering the condition of the product.	Discovering the level of nonconformances.	Testing, software quality assurance, inspections, reviews.
	Assuring the achievement of quality.	Quality control gating.	Product quality audits, go or no go decisions to release or proceed.
Preventing poor quality from occurring.	Quality basis definition.	Efforts to define quality, set quality goals, standards, and thresholds. Quality trade-off analysis.	Defining release criteria for acceptance testing and related quality standards.
	Project and process-oriented interventions.	Efforts to prevent poor product quality or improve process quality.	Training, process improvements, metrics collection, and analysis.

Table 1. *Cost of software quality model categories.*

- How much does poor software quality cost?
- How much does good software quality cost?
- How good is our software quality?

Once the answers to the above questions are factually known, then

- Quality costs can be compared to overall software production costs and software profits.
- Quality costs can be compared to benchmarks and norms.
- Deeper analysis can lead to actions taken to improve the competitive situation.
- The bottom-line effect of quality programs and improvement actions can be measured.
- Previously hidden costs related to poor quality become visible.
- The economic trade-offs involved with software quality become visible, thus leading to better decision making.

Software companies concerned about both product quality and economics can successfully apply cost of quality principles to their software developments, as shown in the remainder of this article. CoSQ is the framework used to discuss how much good software quality and poor software quality costs.

Adapting Cost of Quality Principles to Software

The principles behind the modern cost of quality (CoQ) concept were derived for manufacturing applications and can be found in the works of J.M. Juran [3].

In conventional quality literature, P.B. Crosby [4] asserted that "it is always cheaper to do the job right the first time." However, this statement must be reconsidered with respect to software development. Software is, to borrow a metallurgical term, inherently malleable, capable of being readily shaped, formed, and reworked to alter or refine its function, its quality, or even its purpose.

Malleability is an important reason to develop technical solutions in software rather than in hardware. It allows business and technology to adapt to rapid changes in the world, revising objectives and requirements to address new opportunities as they arise. Both customers and producers have come to rely on software's ability to accommodate changing requirements, giving new meaning to "do the job right." A static sense of what is right cannot be presumed during the lifecycle of many software development projects, thus giving rise to nonmanufacturing-oriented lifecycle models for software, e.g., spiral, incremental, and evolutionary. This additional dynamism strongly influences the economics of the software lifecycle and therefore the application of quality cost principles to software. Establishing and maintaining a baseline definition of what is acceptable quality becomes a key component in the new model for CoSQ.

Applying CoQ Principles to Software

The basis for the new model of CoSQ is the accounting of three types of costs:

- Those incurred due to a lack of quality.
- Those incurred in the appraisal and achievement of acceptable quality.
- Those incurred to prevent poor quality from occurring.

Costs due to a lack of quality are further divided into costs of internal nonconformances and costs of external nonconformances. Costs of achieving quality are further divided into appraisal costs and assurance costs. Prevention costs are found both in the development cycle and in organization-wide activities, such as process improvement and metrics collection and analysis, as well as in quality basis definition and management.

Table 1 provides definintions of the three main CoSQ categories with the next level of breakdown for typical subitems. The term "nonconformance" means a

deviation in one of the software work products with respect to understood objectives, requirements, constraints, or standards. A more detailed taxonomy of CoSQ categories is available in [5].

An Economic Model of Software Quality Costs

There is no validated economic theory of software quality in existence today, which clearly makes it a ripe subject for multidisciplinary research. The software community currently uses a cost of quality theory borrowed from manufacturing, which is exhibited in Figure 2.

Once quality is defined, the costs of achieving quality (costs of conformance) and the costs due to lack of quality (costs of nonconformance) have an inverse relationship to one another—as the investment in achieving quality increases, the costs due to lack of quality decrease, a relationship shown in Figure 2. The quality metric for software is usually a defectiveness level, such as number of defects per system (or part). In traditional CoQ models, the total cost of quality (TCoQ) has a point of diminishing returns, a minimum prior to achieving 100 percent of the quality measure. Current research is investigating whether the law of diminishing returns applies to the CoSQ in all cases.

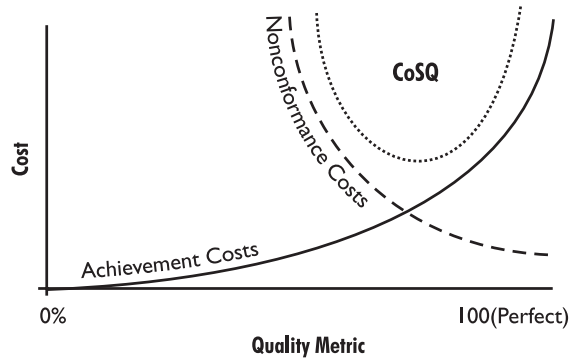
As an industry, we have collected little data about CoSQ that could be used either to challenge or validate this theory. The little we do have suggests that this economic model may not account properly for the dynamics of software development, since perfection is either not a goal or is a quickly moving target.

The NASA space shuttle software program collected and reported on data in this area. In this case, failure-free software is the goal for much of the software system that flies the shuttle. As shown in Figure 3, Keller and Rhone [6] were able to show the increasing cost of achieving extremely high reliability in the mission-critical parts of the flight-control software of the shuttle.

CoSQ: Data Found in the Open Literature

Although the costs of software quality assurance and process improvement have been a topic of concern for over 20 years [7], extremely limited data has been available in the open literature that discusses CoSQ. The main sources to date are a Price Waterhouse study [8], my report [9], S.T. Knox's article [10],

Figure 2. *Economic model of software quality costs.*



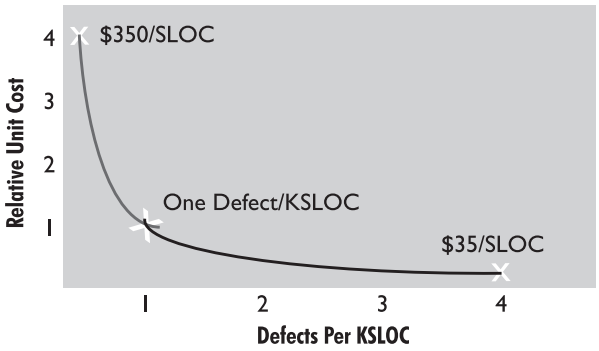


Figure 3. *The cost of ultra high reliability in the shuttle software.*

and the Raytheon efforts [11], which all discuss trends in software rework costs.

The Price Waterhouse study analyzed the costs and benefits of software quality standards from a survey of 19 United Kingdom software suppliers. The study estimated the cost of a quality control effort (prevention and appraisal costs) to be 23 percent to 34 percent of development effort. The study also estimated failure costs at 15 percent of development effort for a total cost of software quality (TCoSQ) of 38 percent to 49 percent of development effort. It must be noted that this study excluded the costs of unit testing and rework because the suppliers could not separate these costs. With increases in the estimates to account for this oversight, TCoSQ in a software organization with a quality system can range from 40 percent to 55 percent of development costs with a conformance costs to nonconformace costs ratio from 1.5-to-2.

Based on the results of my study of Lockheed projects at various Capability Maturity Model (CMM) levels along with anecdotal and quantitative data collected in the mid- and late 1980s, I predicted the relationship of Software Engineering Institute (SEI) CMM-based process maturity level to typical rework rates and quality levels that could be expected [9]. Table 2 is a slice of that presentation.

Due to the SEI CMM process maturity movement, we have an aggregation of the payoff data that has been collected as a result. See [12] for more information on how software process maturity is related to CoSQ, software defectiveness levels, and other measures of success.

R. Dion [13] used the CoQ model as one means to interpret the results of improvement initiatives undertaken at Raytheon Electronic Systems (RES). Recently, T.J. Haley, et al. [11], updated this study. Using the results of tracking 15 projects, they recorded significant results in a little over three years. In the Level 1 stage, RES's CoSQ fluctuated between 55 percent and 67 percent of total development costs, and when reaching Level 3 process maturity, their CoSQ had dropped to approximately 40 percent of total project cost. The ratio of conformance to nonconformance costs was 1.5. By 1996, this organization's TCoSQ was approximately 15 percent of development costs, and the rework due to both internal and external nonconformances has been reduced to less than 10 percent of development costs.

Process Maturity (characteristic)	Rework (percent of total development effort)	Product Quality (defect density)
Immature	>= .50	double digit
Project Controlled	.25-.50	single digit
Defined Organizational Process	.15-.25	.X
Management by Fact	.05-.15	.0X
Continuous Learning and Improvement	<= .05	<.00X

Table 2. *Process maturity, rework, and quality results.*

Knox made similar predictions about CoSQ and rework expectations across the levels of the SEI CMM (Figure 4) [10]. Starting with the (TCoSQ) at 60 percent of development costs (based on two industry figures) for CMM Level 1 organizations, Knox used manufacturing experience to hypothesize that CMM Level 5 organizations can cut this CoSQ by about 67 percent. He then rationalized the four component costs at each CMM level. Knox's model predicted that a CMM Level 3 organization would have a TCoSQ of 50 percent but with a conformance to nonconformance cost ratio of .5. It appears that Knox's model may be a fair predictor of TCoSQ for maturing software organizations but that actual conformance costs are much higher and nonconformance costs much lower than what the model predicts.

Typical manufacturing CoQ, ranging from 5 percent to 25 percent of company sales, contrasts significantly with CoSQ. With the present state of software engineering practice, we can expect CoSQ to range from 10 percent to 70 percent of development costs. Even accounting for the margin between production costs and sales, CoSQ appears to be roughly twice manufacturing CoQ. Also, the optimum manufacturing CoQ is often in the range of 95 percent to 100 percent of conformance to quality standards. The open literature lacks data for CoSQ as a function of conformance to quality, but the above data suggests that software producers have yet to reach such an optimum.

Elements of a CoSQ Program

There are many possible ways to apply the CoSQ approach. To date, CoSQ techniques are only being used after the fact to document the return on investment (ROI) for software improvement initiatives because executives want to know that there is a payoff from the upfront investments. This type of CoSQ application is expected to accelerate as more process improvement programs take off.

Other ways in which the CoSQ approach can be used are to

- Provide a basis for budgeting the quality management and assurance functions.
- Identify specific quality improvement candidates through causal analysis.
- Compare proposed process improvements and identify the most cost-effective ones.
- Provide a (one) measure to compare the success of various projects.
- Reduce the quality costs on a particular project by altering the process prior to or even on site.
- Determine the potential cost and risk impact of specific quality trade-off decisions on specific projects.
- Determine a company's potential legal exposure due to customer-experienced defects.
- Provide cost data to demonstrate the relationship of employee efforts to the bottom line.

CoSQ Programmatics

Several points can be made with regard to the programmatic aspects of measuring and using CoQ information specifically for software development organizations. These are

- Initiating a CoSQ effort.
- Accounting and gathering the quality cost data.
- Gathering the quality metrics.
- Presenting the results.
- Improving the CoSQ program continuously.

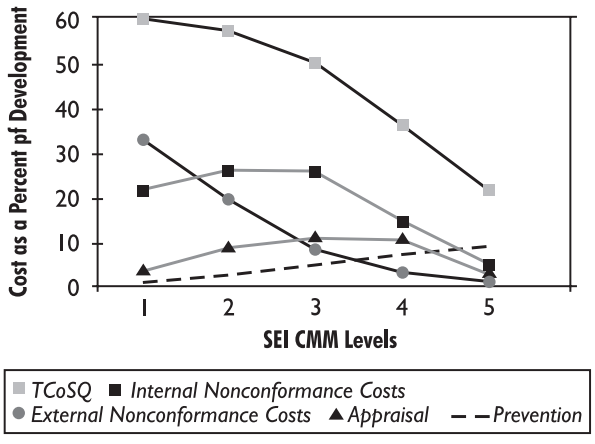
Initiating the CoSQ Effort

Convincing management of the value of tracking CoSQ may be the initial hurdle one encounters in using this technique. There is a modest upfront investment required to educate those to be involved.

Initially, rough estimates of software quality costs may suffice well for several reasons.

- Usually, the largest CoSQ costs can be estimated readily from time and activity reports, so the expense of data gathering can be limited until its value is demonstrated.
- Controlled, scientific studies are unlikely, and incomplete data can suffice in beginning a software cost-benefit analysis.

Figure 4. *Cost of software quality and CMM level.*



- The published data indicates that the quality cost difference between improved and unimproved organizations is quite large.
- The primary purpose of the initial CoSQ effort will be to show the opportunity for cost savings.

The best advice is to use the "keep it simple" principle in starting a CoSQ initiative.

Accounting

Gathering quality cost data assumes that costs have been accounted using task categories that can be summed into the four major categories of quality costs. Many software organizations track costs in a manner amenable to quality costing, but many others do not. In the latter case, a preliminary step of defining and installing such a chart of accounts is required. A sample of such a chart of software quality costs can be found in [5]. The quality categories in a software organization's chart of accounts must be tailored to reflect its software process. To realize the full benefit of CoSQ, it must also allow for the addition of continuous improvement tasks.

In the best cases, quality costs can be taken directly from departmental accounting (salary and expense) reports. In other cases, it may be necessary to resort to basic accounting and engineering records, such as schedules, time reports, defect reports, and purchasing records. In the worst cases, one may fall back on interviews with members of the software organization to construct estimates of each quality cost category. Exceptions are in the external failure category.

One of the pitfalls of a CoSQ program is "controversial cost categories." Usually, the question is about which costs are normal operating costs and which are quality costs. An example would be the cost to produce a project management plan. Although this plan is produced for the sake of managing a project's expenses and schedule, it also influences product and process quality. In this case, it is helpful to keep in mind the following points.

- The trend among quality specialists has been to view quality costs as those incurred to directly prevent, appraise, and address the nonconformances of *poor quality*.
- Arguments over controversial categories have been known to sabotage cost of quality programs.
- The largest quality costs are those that are most easily discerned, for example, reviews, software quality assurance, testing, and rework; therefore, it is often safe to exclude controversial categories without unduly affecting the TCoSQ.
- Consistency throughout a CoSQ program is more important than thorough inclusion of quality costs because consistency allows for clear identification of improvements and candidates for improvement.

Concerns may also arise as to how quality costs should be categorized. Again, consistency is important. For example, the costs associated with formal inspections (peer reviews) can be treated as prevention costs rather than as appraisal costs. This is a matter of interpretation, depending on when a work product is considered ready for appraisal. Although manufacturing inspections are conducted on pieces after they are produced, in software production, inspections may be incorporated into the production process. For documentation, this means that a document is not complete until it has undergone a peer review and has been revised. The same is true for code, especially when code inspections precede unit testing—clearly an appraisal activity.

Quality Metrics Collection

With regard to measures of quality, the CoQ has been used primarily with a fundamental approach to quality; that is, defect rates (manufacturing) or service problem reports (service industries) rather than broader approaches that would take into account factors such as usability, testability, maintainability, and so forth. The fundamental approach has the advantages of straightforward measurement and ease of understanding. It also allows comparison of dissimilar products. Furthermore, if failure costs are collected in a defect tracking system, the most expensive defects can be identified for root cause analysis [14]. This discussion recognizes that most software producers take a fundamental approach to quality, concentrating on defect measurement, prevention, and removal.

Defect density is a good metric to start with measuring CoSQ improvements; specifically, CoSQ can be plotted against defect density at the completion of system testing. This metric may be obtained from defect reports during alpha and beta tests and for a period, e.g., six months, following product release. Better yet, it may be generated statistically based on post-release defect reports for previous products from the same organization. Robert Stoddard and John Hedstrom [15] offer a recent example of this approach using Bayesian statistics in a defect-prediction model. External failure costs can be estimated from the defects-at-release metric.

Presenting CoSQ Information

The relationships that have the greatest impact on management are

- Quality costs as a percent of sales and profit.
- Quality costs as a percent of total development costs.
- Quality costs compared to the magnitude of the current problem.

Showing CoSQ as a percent of total development costs is appropriate to software for several reasons. First, sales and profit may not have a direct relationship to the cost of a software product since software pricing is often dictated by market forces. Second, all but a small percentage of software development costs can be measured in labor hours, so the costs can be readily shown in either hours or dollars. Third, the state of the art in software development is such that comparing quality costs to development costs illustrates the magnitude of the current problem.

Though quality costs as a percent of development costs can show significant effects of improvements, this ratio does not reveal the optimum cost of quality. The optimum can be seen when quality costs are shown as absolute costs against a quality measure. Plotting CoSQ costs against a quality measure, such as defect density, reveals trends in an organization's quality processes, e.g., in [16].

Improving the CoSQ Program

Based on initial usage of CoSQ, organizations should expect to encounter difficulties in the following areas.

- When and how the CoSQ data is gathered, analyzed, reported, and used.
- How the approach clashes with other approaches that are already in use, e.g., existing work breakdown structures that do not map easily to CoSQ categories.
- How the CoSQ model is defined at the detailed levels.
- How the approach is implemented consistently in the organization.
- How CoSQ is used for root cause analysis.
- How CoSQ is used to stimulate improvements.

These difficulties can be overcome with appropriate training and coaching.

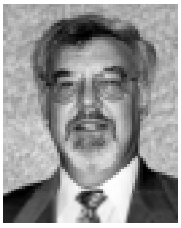
Feedback on the usefulness of the CoSQ data presented can guide how the organizational CoSQ program should be evolved over time. The lessons learned from trials and early adopters will be invaluable. The technology to support CoSQ will emerge quickly in response to the needs as they grow, once a consensus on the CoSQ model is reached. Many of the Total Quality Management and CoQ tools available for manufacturing can be adapted for use in software. CoSQ tools appear to be a significant market opportunity yet to be explored.

Conclusion

CoQ is a proven technique in manufacturing industries for both communicating the value of quality initiatives and indicating quality initiative candidates. CoSQ offers the same promise for the software industry but has seen little use to date. Initial uses of CoSQ show that it can be a large percentage of development costs—60 percent or higher for organizations unaware of improvement opportunities. CoSQ has demonstrated its value in measuring the ROI of a software improvement program, as in the RES case.

CoSQ is a technique that is most useful in enabling our understanding of the economic trade-offs involved in delivering good-quality software. Applying CoSQ in your organization requires a small investment that pays off handsomely in your increased understanding of the complexities and hidden issues involved in the delivery of quality software. The proliferation of CoSQ will help eliminate the debilitating effects of poor software quality. ◆

About the Author



Herb Krasner, president of Krasner Consulting since 1991, has almost 30 years experience in the software engineering profession as a practitioner, researcher, and teacher. He is a master lead assessor, having performed over 40 CMM-based appraisals in the last 10 years since being certificated by the SEI. He has also been involved as a subject-matter expert in a number of computer-oriented legal actions. He is the founder of the Austin Software Process Improvement Network, chairman emeritus of the Software Quality Institute at the University of Texas and has been chairman of or keynote speaker at several international conferences. He also teaches the body of knowledge for the ASQ Certified Software Quality Engineer Program. He has frequently published and presented his work in many professional forums.

Krasner Consulting
1901 Ringtail Ridge
Austin, TX 78746
Voice: 512-328-4264
Fax: 512-328-3260
E-mail: hkrasner@cs.utexas.edu

References

1. *Software Quality Matters*, http://www.utexas.edu/coe/sqi.
2. Davis, A., Editor's Column, *IEEE Software*, December 1997.
3. Juran, J.M. and Frank M. Gryna, *Juan's Quality Control Handbook*, 4th ed., McGraw-Hill, New York, 1988.
4. Crosby, P.B., *Quality Without Tears*, McGraw-Hill, New York, 1988.
5. Campanella, J., ed., *Principles of Quality Costs*, 3rd ed., American Society for Quality Control, Milwaukee, Wis., forthcoming, 1999.
6. Krasner, H., "A Case History of the NASA Space Shuttle Onboard Systems Project," SEMATECH Technology Transfer Report 94092551A-TR, Oct. 31, 1994.
7. Alberts, D.S., "The Economics of Software Quality Assurance," *National Computer Conference 1976*, pp. 433-441.
8. Price Waterhouse, *Software Quality Standards: The Costs and Benefits: A Review for the Department of Trade and Industry*, Price Waterhouse Management Consultants, London, 1988.
9. Krasner, H., "Self-Assessment Experiences at Lockheed," *Proceedings of the SEI/AIAA Software Process Improvement Workshop*, Chantilly, Va., Nov. 8, 1990.
10. Knox, S.T., "Modeling the Cost of Software Quality," *Digital Technical Journal*, Vol. 5, No. 4, 1993, pp. 9-16.
11. Haley, T.J., "Software Process Improvement at Raytheon," *IEEE Software*, November 1996, pp. 33-41.
12. 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," *The Elements of Software Process Assessment and Improvement*, IEEE Computer Society Press, 1998.
13. Dion, R., "Process Improvement and the Corporate Balance Sheet," *IEEE Software*, July 1993, pp. 28-35.
14. Mandeville, W.A., "Software Costs of Quality," *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 2, 1990, pp. 315-318.
15. Stoddard, Robert and John Hedstrom, "A Bayesian Approach to Deriving Parameter Values for a Software Defect Predictive Model," *Proceedings of the Sixth Annual Conference on Applications of Software Measurement*, Oct. 2 – Nov. 2, 1995, pp. 323-346.
16. Houston, D., "Cost of Software Quality: Selling Software Process Improvement to Managers," *Software Quality Journal*, forthcoming, 1998.