Annals of Software Engineering 1(1995)119–139

Software quality management through process and product modeling

Robert C. Tausworthe*†

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, USA

This paper presents a simple cost-based method for managing software quality during the development process. The technique may apply conceptually to the management of quality for other processes, as well. A unifying metric for quality is defined to be the ratio of the cost expended at a given point in time to that which will be required to satisfy all requirements placed on quality attributes, such as correctness, reliability, etc. Unit quality then corresponds to a system that fulfills all of its quality attribute requirements. The paper indicates the need for developing cost-versus-attribute relationships for all quality factors of concern.

1. INTRODUCTION

Quality is an intrinsic, multifaceted characteristic of a system. We perceive it as the composite of features and characteristics of a system that bear on its ability to satisfy given needs, together with the assessment of the degree to which the system possesses individual desired attributes. Evaluations of system quality by analyses, metrics, tests, reviews, and usage, with qualification against objective standards, are indicators of the levels of quality achieved.The creation of a high-quality system—one that will meet the composite needs and requirements of user and operational communities—demands that the pertinent quality factors be specified, designed, engineered, and built into products from the beginning, so that the perceptions of the end product are satisfactory.

We judge the composite quality of a system by the extent to which its range of attributes fits the needs and desires of its customers and users. Metrics of quality are manifest in such quantities as operational performance, test effort, time to failure, anomaly history, anomaly criticality classification, time to restore service, effort to remove anomalies, amount of training necessary for maintenance personnel, document consistency and completeness, traceability among products, and conformance to product

*The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

† E-mail: robert.c.tausworthe@jpl.nasa.gov

workmanship standards. The correlation of these measures with other product attributes (e.g., cost, size, complexity, functional type, amount of reuse, and new technology requirements) and environmental attributes (e.g., production work place, computer resources, level of technology, and staff expertise) potentially enable the statistical determination of general trends and dependencies among attributes, which then become useful quality indicators within appropriately defined ranges of measured validity.

Any shortfall in a quality attribute from its required value is a *quality defect.* We achieve quality through planning and building products with specified attributes, by defect prevention, and, failing this, by defect detection and removal. The process of defect prevention and removal, i.e., quality engineering, is not the subject of this paper. The influence of management on this process, however, is.

The advent of Total Quality Management (TQM) [Deming 1989] presented the software industry with a challenge to execute major changes in attitude and direction to achieve "quality-first products". The TQM view is that we improve quality in products by focusing on customer needs and streamlining the production process to eliminate waste. TQM considers quality as the antithesis of loss, or waste, associated with a product due to deviations of its characteristics from target values. High quality in products corresponds to low losses in production processes. The works of Taguchi [Elsayed *et al.* 1988] and others [Walton 1988; Phadke 1989] in robust design produced a methodology and set of tools for statistical optimization of the production process, so that high-quality products could be mass produced more quickly and at lower cost.

Some skeptics maintain that the TQM approach cannot effectively be applied to software projects because their products are unique; in reality, however, TQM principles are consistent with good software practice. Recognition of "quality" as the absence of "loss" distinguishes processes in terms of their consequences – errors, awkwardness, embarrassment, penalties, loss of reputation, suffering, damage, ruin, etc.

Quality, cost, and risk are intricately related. The cost expended to forestall risks is thus a natural standard for judging quality. Cost is a unified and universal measure that applies equally to measurement of customer dissatisfaction, defective products, operational excesses, inferior utilization, and poorly used human resources.

Pall [1987] claims that quality in the United States has not necessarily been on the decline; contrarily, in most sectors of our industry, it has continued to improve. But the rate of improvement has seemingly not been sufficient to satisfy increasingly demanding requirements and expectations of customers. He contends that this is due to the dissection of the quality process into individual elements, separately optimized. Rather, a more global optimization of the process, he states, is needed. Quality, he argues, is not so much a technical problem as it is a management problem.

2. MODEL BASED MANAGEMENT

The term *model based management* here refers to supervision of software development and/or maintenance efforts assisted by quantitative interrelationships

among metrics applied to the development process and its products. In truth, every manager already uses model based management in one form or another, some portions of which may be formal; but, more likely, they are informal mental intuitions gained through training and experience.

Mathematical models and metrics for interrelating complexity, cost predictability, and quality are still in their adolescence, and remain experimental. Formal models, as currently exist, are controversial in matters of validity, accuracy, and application. They mostly address cost and reliability, and none has emerged to be definitively superior to the others in its class [Lyu 1991].

However, we know that certain metrics do correlate, albeit imperfectly, with system performance and development status. These statistical measures not only provide a tenable degree of quantitative information for improved management, but, if collected, they also provide a database for future project planning and model based management research.

Modern managers, meanwhile, do plan definable development processes, adhere to adopted standards, use assistance tools, create effective organizational structures, promote particular staff objectives, etc., as matters of good practice. Such infrastructures are forms of process methodologies that are becoming increasingly more articulated and assessable.

In its maturity framework, the Software Engineering Institute (SEI) [Humphry 1987] identifies five levels of process maturity and two stages of technology advancement that characterize the capabilities of enterprises. At the optimized maturity level (Level 5), enterprises have not only achieved a high degree of control over their processes, but they have also placed a major focus on improving its operation. They routinely apply quantitative, model-based management principles by making sophisticated assessments of quality and costs from data gathered during the process; they also execute comprehensive cause analyses and prevention studies to avert poor quality or cost performance. Even the second SEI maturity level demands process repeatability in an enterprise.

The SEI Capability Maturity Model (CMM) evolved from then-current best practices and a theory of what future best practices would be like. That theory has not yet been fully validated quantitatively; however, the Department of Defense [Mosemann 1993] estimates that 64% of its software project documentation costs could be avoided by only using contractors having been evaluated as CMM Level 3 or higher, who have the relevant domain knowledge, and who have a satisfactory skills matrix.

The "universality of management" concept launched by Koontz *et al.* [1984] is a non-quantitative model in which the fundamental functions and activities of management are the same, regardless of the management positions within the organization or the enterprise managed. The detailed application of these fundamentals, however, does change depending on position, enterprise, and domain. Thayer [1988] identifies and describes a basic set of software and project management activities and tasks

necessary to successfully manage a software engineering project. He also depicts the introduction and rise in utility of submodels, which include scheduling and monitoring techniques (e.g., PERT and CPM), organizational structures (e.g., project versus matrix), motivational methodologies (e.g., Theories X-Y-Z, MBO, and Maslov-Need-Hierarchy), and control disciplines (Configuration Management).

These models, and many others used today by software projects, are chiefly oriented toward procedure and form, and are not particularly quantitative. Still others, such as those found in [Gilb 1977] and [Perlis 1981], are more quantitative, but lack unity. Further, there is controversy in how measurements relate to assessing the quality of processes and products. The fact, that most enterprises so far rated by the SEI lie in the lowest two maturity levels, is a fair indication that model-based quality management has not yet penetrated deeply into the software industry.

But as the industry matures to the point that its processes are repeatable, such methods will surely evolve, for any process that is repeatable becomes predictable, and thus can be modeled. The mere fact that these methods may not exist today should not deter us from investigating the form, utility, and benefit that they will someday have.

Model-based management depends on forecastability of events, quality attributes, and resources. Traditionally, forecasting human behavior has been successful only in structured situations, or in making near-term projections, or in sketching general trends. Hence, to foster success, our quality-based method will have to impose structure on the development process, primarily require only near-term forecasts, and display basic trends that can continually improve the method.

This paper introduces two simple, but novel concepts of quantitative quality management based on operations research techniques. Admittedly, this approach is yet in the straw man theory stage. Much work lies ahead before sufficient maturity in the method, technology, and usage come about that could qualify the method for industrial use. However, it is proper now to introduce the basis of unified quality metrics, to discuss how the metrics could be used, and to forecast what kind of measures will be required if total quality is to be quantitatively managed.

We can illustrate the method with hypothetical examples; but we cannot present case studies, because no body of supporting empirical evidence has yet been developed. It is thus fair to view this approach with some skepticism. Technical literature traditionally publishes untried methods and models for public review and further development. The application of the proposed method will require both cultural and technological changes that will not occur overnight.

The quality management method presented here springs from fundamental considerations of the underlying natures of quality, software, and the development process. Real-life processes are flexible, intricate, and complex. Quality attributes, schedules, and costs are interrelated by product requirements, available funding, the chosen life cycle model, process controls, personnel aptitudes, usage of tools and technology, company organizational structure, customer/user interfaces, and work friction.

With respect to the role models play in management, we recognize that they are only tools that provide advice. They generally cannot be held accountable or liable when they fail. Human performance is stochastically variable, and all methods, no matter how carefully and accurately calibrated, will certainly and expectedly produce uncertain estimates. All advice, from humans as well as models, should be weighed in respect to its perceived uncertainty and bias.

3. QUALITY-COST RELATIONSHIP

Belady [1977] mused that it was "discouragingly difficult to construct a [quality] index which measures not only the end product but also the process leading to it". He foresaw that such a measure, if it existed, would constitute a model of how software evolves from its requirements to an operating, maintained program. Mere charts of disparate, unrelated measurements of the sequential phases could not be considered sufficient information for successful management.

Crosby [1979] contended that "quality is free", but that "no one is ever going to know, if there isn't some sort of agreed-on system of measurement". He classified the "costs of quality", or COQ, as (1) the extra costs that are expended to do the job right the first time, (2) all costs involved in doing the work over, and (3) operational liability costs. Crosby argued that the costs to achieve a given level of quality do not increase when expenditures are concentrated in the first category—the "extra costs" perhaps thought necessary to do the job right in the first place are, in fact, not extra in the overall cost sense. For while quality *competes* with productivity during development, if operations costs and utilization benefits are factored in, increased quality tends to *increase* overall productivity. The cost of quality thus depends on the length of the life-cycle and the impacts of defects in usage. If life is long, quality is free, although development costs are perhaps higher.

Boehm [1978] and Gilb [1977], attempting to capture the elusive essence of software quality that will validate Crosby's hypothesis, have developed a comprehensive arsenal of metrics. Taken together, these metrics quantify numerous factors that comprise the concept of quality. Their approaches are similar, although they differ somewhat in structure. Together, they aggregate about 175 specific elemental measures into some 41 composite quality factors.

One problem with applying Crosby's definition of COQ is the difficulty in measuring the "extra costs" for quality. The evaluation of extra expenses for quality requires that costs (1), (2), and (3) above be separately planned, managed, and accounted. But these costs are frequently not measurable without special, separate efforts. Those "extra costs" actually incurred in building a superior product are seldom directly observable or freely furnished; they are typically estimated after-the-fact from the recollections of those having done the work. Project management systems and enterprise accounting systems seldom have provisions for measuring just when developers are doing the job correctly the first time, and when they are doing it incorrectly the first time.

Another view [Stamm 1981] is that quality is not merely what a Quality Assurance effort provides, but what the entire project staff produces. Software projects seek to expend their budgets to produce and assure a set of products whose function, performance, and other quality attributes are acceptable. The entire staff is responsible for the product set and the quality attained. Therefore, he argues, the *total cost* to develop products to their delivered and maintained state is the *true cost of quality*, TCQ.

This paper adopts the TCQ view: all development expenditures are made for the sake of quality, including those for specification, design, implementation, integration, validation, and installation/delivery. Depending on a particular project's scope, management of quality may not end upon delivering the required functions and performance, but may extend into the maintenance and support phases, as well.

Realistically, when customers require a quality factor in a system, there is a cost required to develop it. We can build slow, failure-prone, cumbersome systems sometimes at considerably less cost than fast, reliable, easily used ones. We may thus expect the costs to build a system to increase as a function of the required level of quality. In developing a system, many attributes of the process and products compete for project resources. Curtis [1981] notes that probably no software project will ever be able to stay within its allocated resources and maximize all quality factors; rather, it will be necessary for quality attributes to be weighed according to the nature of the system, the needs of the users, and the resources available.

Once the proper value of a given quality factor has been agreed upon, then the extra cost of achieving that required quality may be free, as proposed by Crosby, whether incurred during the initial production cycle or retrofitted into the system at a later time. There may be penalties and associated risks in a system delivered with a prescribed level of quality absent. As a result, the quality-related cost will be incurred, either in expenditures to update the quality, or in loss of benefits, or in operational risks. The rational approach to building a superior system, then, is to expend the resources necessary to achieve the required quality attributes in the most cost-effective manner.

Besides competing for resources, quality attributes may be highly interrelated. Work expended to positively service one attribute may negatively impact another. The classic case of this is that work performed to meet performance objectives frequently results in highly optimized, complex code. Conversely, however, work performed in servicing some attributes may benefit others. For example, costs expended toward portability tend to abet efforts in reusability and interoperability. Clearly, managers must know how to apply resources to the various required quality attributes in order to conduct the software process successfully.

4. QUALITY MEASUREMENT

The perception of quality varies by individual and over time, but, on the average, it is governed by supply and demand economics and may be assessed as a balance

between value, measured by satisfaction of needs, and worth, measured by expenditure of precious resources.

On this basis, the method presented in this paper presumes that, **properly managed, the overall cost of a system measures the level of the quality it has paid for.** This is a TCQ theme that applies to customer, user, and sponsor points of view, where needs for quality often may be mitigated by affordability, and the cost of that quality becomes a competitive issue.

For example, one *expects* that an automobile selling for $50,000 will have been built to meet quality (and profit) goals five times more difficult to reach (for that manufacturer) than those of a car selling for $10,000. Several cars, each selling for $10,000, may have different sets of quality attributes, as well as different levels of shared qualities, depending on the goals and productivities of the manufacturers. Owners willing to spend no more than $10,000, who buy and find satisfaction in one or another of these, tacitly acknowledge that their quality goals and affordability constraints have been met within their (perhaps unarticulated) requirements for automobiles. In effect, they have optimized their latent quality-attribute-versus-affordability function.

Those who are unsatisfied with all $10,000 cars must spend more to achieve the higher levels of quality they desire. Those unwilling to spend more become potential customers of entrepreneurs who can better deliver satisfaction at the given price, if that is possible. Those buying cars anyway, but still having unsatisfied needs, have thoughtfully bought the highest quality they can afford. There is then a "quality gap" equal to the difference between the price of an acceptible-quality car and the price of the car actually bought.

Similar expectations of quality persist whether software is developed or purchased off-the-shelf. In the case of commercial database management systems, word processors, and other software items on the market, various packages are available in a range of prices and possessing various mixtures of features and performance characteristics, desired and abhorred. The duty of the buyer is to match functional characteristics and quality features against the intensity of need for them, and to determine the greatest affordable match.

Statistically speaking then, in a free market economy, quality corresponds to price. The amount of quality that one can afford is fixed by economics, whereas the particular attributes of quality purchased are determined by cost effectiveness.

Whether or not software customers and users are able to articulate their needs for quality attributes, it is, nevertheless, possible to express quality attributes of an information system in absolute, rigorous terms, such as mean time to failure, percentage of portable code, achieved processing speed, and cost to develop. The units of these measures are hardly conformable; however, they do serve as figures of merit[1] that distinguish the system.

[1] *It gives us strangely little aid,*
But does tell something in the end.
– Robert Frost

To make quality assessment less subjective than individual perceptions of satisfaction, we will require that target values of quality attributes be articulated and tabulated in testable form, whether met or not, and, if not, assessable at any time as to the resources required to remediate and fulfill the requirement.

All projects measure expended costs, if nothing else. They must also predict, or otherwise assess, the runout costs for all remaining work. They normally do this at key points in the development process, for costs must be managed. Managers are thus perhaps more aware of costs than of any other project metric (except for schedule, perhaps, in some cases).

Software developers who can relate quality attributes to cost have the means to manage the quality of the products they build. They can build custom products for customers who can articulate their quality needs, and they can build product lines on supposed needs of their potential customers, which will be validated in sales and market share.

Some quality attributes may be deemed more meaningful when expressed in relative terms or in normalized forms. These measures tend to relate the degree to which a given quality attribute is present in a particular entity. For example, if one defines

$$\text{reusability factor} = 1 - \frac{\text{cost to reuse}}{\text{cost to redevelop}}$$

for a particular object, then this factor serves to indicate the effectiveness with which this object can be incorporated elsewhere. A score of unity expresses high reusability, indicating zero cost to reuse. A score of zero or less indicates no reusability, for the cost to reuse exceeds the cost to redevelop. The reusability indicator thus relates this quality attribute to a measurable (and forecastable) cost parameter.

Other quality factors may also be related directly to cost, as suggested by the entries in table 1. Some of these may be readily measurable, while others may perhaps be inferred through factor analyses of measured relationships among composite quality and overall cost. Guidelines for use in evaluating software product quality may be found in [ISO 9126].

This paper proposes the usage of cost as a causal measure of quality, in that it directly influences each quality attribute in a dominant way: a cost applied to servicing a given attribute will tend to improve that attribute, and, moreover, that attribute will not generally tend to improve without it.

The use of cost as a unifying quality metric depends on several supplementary hypotheses and assumptions, among which are

(1) it is possible to articulate and to measure or quantitatively assess the status of all quality attributes of interest, and

(2) it is possible to relate quality status to a forecast of resources required to achieve the specified levels of quality.

Table 1

Major quality factors and cost measures.

Factor	Attribute indicators
correctness	Cost to meet functional requirements, remove faults, process liens, and establish workarounds
reliability	Cost to produce required probability of operating successfully
efficiency	Cost to produce specified performance margin
security	Cost to produce required probability of only authorized access
integrity	Cost to produce required probability of damage due to improper access
usability, operability	Cost to reach specified training, familiarization, input prepration, and output interpretation cost goals
survivability	Cost to produce required probability of continued perforance when a portion becomes inoperable
maintainability	Cost to produce given repair-cost/fault ratio
testability, verifiability	Cost to produce given test-cost/total-cost ratio
flexibility	Cost to alter a given function
portability	Cost to port to a new host
reusability	Cost to produce given ratio of reuse-cost to redevelopment-cost
interoperability	Cost to couple and operate with another system
expandability	Cost to add new requirement
availability	Cost to produce required probability of accessibility for given purpose
adaptability	Cost to adapt to changed host system characteristic or different user interface or application

Even if we cannot do some of these things now, the article of faith of this paper is that, with measurement and study, they become possible in the future.

At each decision point, managers assess and judge whether quality attribute values are sufficient or deficient for that point in time. They use this assessed state of quality, together with the project history, the profiles of expended and planned resources, and causal indicators to estimate the remaining outlay of resources necessary for attaining quality goals. A properly conducted quality metrics program provides the history and current data upon which the evaluations operate; confidence grows as the methods improve.

When the cost to reach *all* the required quality attribute levels is too high or unjustifiable, quality in some attributes must be sacrificed to the benefit of others. We thus seek to establish balanced quality attribute costs by developing a benefit indicator in the form of weighting factors applied to individual indicators based on relative importance, value, priority, risk, and performance goals. We may determine individual attribute requirements and goals from physical needs (e.g., function, performance, safety, and risk) or associated economic and corporate goals, such as are inherent in maintainability, usability, and adaptability. Strategies for attaining balanced quality goals are treated in section 7 of this work.

5. QUALITY BREAKDOWN STRUCTURE

The degree to which runout cost predictions can be made to be accurate translates directly into the certainty in overall achieved quality. Estimates of cost and quality uncertainty are necessary for risk management. Boehm [1981] notes that software cost estimation accuracy tends to improve as projects progress; nevertheless, unreliable cost forecasting and ineffective cost management can lead to inevitable situations in which planned costs have been totally expended, and yet the system has not met its quality objectives. Ineffective cost assessments can lead to the "90% complete" [2] syndrome, in which managers estimate that 90% of the quality goals have been met far before they have sufficient basis to make that judgment.

Fortunately, cost accounting based on earned-value assessment and Work Breakdown Structure (WBS) techniques [Tausworthe 1980] now commonly used in most large software projects, tend to prevent this from occurring. The WBS method challenge is to plan all project costs and resource expenditures as a hierarchy of activities, each with a definite beginning and ending point and definite products, with unit work packages of about the same size and expenditure uncertainty each. The WBS itemizes all budgeted project cost and schedule items (and, perhaps, also contingencies). Activities that may be deemed not to contribute to quality become candidates for elimination from the project plan.

WBS milestones are used as quality evaluation points. Earned values, awarded to each work package (inch-pebble) according to the degree that the planned activity has achieved its technical goals, are used by the project to measure the overall status of achievements. Some resource expenditures, such as management, development support, and company overhead, by their nature, may not produce sharp milestones, and may thus seem only peripherally related to quality. Others may be sharp, but may not be deliverables (e.g., trade-off studies), and may not be directly visible within the deliverable item accounting system. But management and support activities do contribute to the quality achieved in the activities they service; their resource expenditures

[2] The 90-90 rule: the first 90% of a project requires the first 90% of the resources. The remaining 10% requires the other 90% of the resources. (*A not-always humorous software engineering adage.*)

may be accounted as a tax to each serviced activity, or may be separately accounted as prorated expended resources.

The WBS earned-value challenge is to assign weights to each of the inch-pebble work packages that reflect the effectiveness of the resources expended, and to account expenditures in terms of earned value. The earned-value metric used in this paper is the relative quality index, a quantity to be defined more precisely in the next section. Briefly, it is the assessment of the degree to which quality goals have been met, as measured by expended versus required costs.

This metric imposes quality requirements on the WBS in the form of criteria for assessing the contribution of each work package to quality goals. Each work package identifies which attributes are being serviced, and at what cost to achieve, in addition to the budgeted cost and schedule allotted to the package. We refer to this quality-oriented WBS as the Quality Breakdown Structure (QBS).

We will discuss methods for properly allocating and accounting costs across quality attributes in a little more detail in section 7. For now, let us investigate the characteristics and metrics that will foster proper quality management, once the QBS and cost-quality relationships have been established.

6. THE RELATIVE QUALITY INDEX

Management metrics are measurements of actual performance parameters, regularly made during a development process. When compared with plans and forecasts, they help to determine the project's health and status. Deviations are indicators of actions that need to be taken and work that needs to be performed over the plan horizon. Even if faulty, actual-versus-forecast and earned-value plots are often very useful in indicating trends, pointing out problems, and improving future forecasting.

At any time, the costs that have been expended or allocated relative to the true cost needed to achieve quality goals is a measure of the degree to which the requirements for quality have been or will be met. A relative cost of unity indicates that the system is of the highest quality required for its intended purposes. The differential between unity and a relative cost less than unity discloses that fraction of the task quality in deficit.

To be more specific, let C_t denote the cost that has been expended up to a given time t, and suppose that an assessment of the current situation now predicts that an additional cost $\Delta\hat{C}$ will be required in order to elevate the system to full compliance with its quality specification. The forecasted total cost of the system is $\hat{C} = C_t + \Delta\hat{C}$; the true cost to reach the desired quality is $C_t + \Delta C$. If the projected cost $\hat{C}$ is within the budgeted cost $\bar{C}$, all is well; if it is not, management action is indicated.

We define the estimated *relative quality index* as the cost fraction thus far incurred,

$$\hat{q} = \frac{C_t}{\hat{C}} = \frac{C_t}{C_t + \Delta\hat{C}} = \frac{1}{1+\hat{\varepsilon}}, \tag{1}$$

where $\hat{\varepsilon} = \Delta\hat{C}/C_t$ is the estimated relative remaining cost, or *current quality deficit.* Similar quantities q and ε based on actual future costs are also defined. Since C is unknown until the quality is actually achieved, we will drop the notational differences henceforth, and rely on context to distinguish between C and $\hat{C}$, and q and $\hat{q}$.

The q index serves as an earned-value measure of quality: it measures the composite satisfaction of quality goals with respect to a set of predefined quality attributes, in terms of true costs to achieve. This is such a simple, but subtle, concept that it deserves more explanation and analysis.

A set of proposed means for reaching stated quality requirements can be compared on the basis of their absolute costs C, and the relative status at any time is indicated by the relative quality indices or relative quality deficits. At point t in time, a manager observes that an actual cost C_t of the budgeted cost $\tilde{C}$ has been incurred, and that $\tilde{C} - C_t$ remains to complete the project. The perceived quality (according to budget allocations) is $q = C_t/\tilde{C}$. The manager wants to know if costs are on track, so a new forecast of the remaining costs required to achieve the specified quality of products is made, resulting in a new runout cost estimate C.

At that point in time when the estimated remaining ΔC will have reached zero, i.e., when the relative quality index has reached unity, the system will have acquired its required quality, and further expenditure will not be warranted.

If the budgeted cost $\tilde{C}$ exceeds the estimated required cost C, then all is well, and no remedial action is indicated. However, when the opposite situation is true, then the manager realizes that quality goals will not be met within the current budget. The ultimate index, only reaching $\tilde{q} = \tilde{C}/C < 1$, is the relative performance shortfall of the developing organization in meeting its contracted responsibility. The customer-specified level of quality will not be achieved under the current plan.

Similar considerations relate to schedule constraints. For illustrative purposes here, however, we shall concentrate on cost constraints only.

Regular assessments of status are a matter of routine in the TQM environment. Managers need to know the quality status of their products in order to be effective. Along with assessments of quality, they also need to estimate the amount of remaining work (cost) required to reach quality goals.

The relative quality index is a natural model-independent combination of quality and cost assessments that is also independent of the scale of the effort. Any methods for assessing costs to achieve specified levels of quality appropriate to the size and complexity of a particular project, formal or informal, may be used. Since cost and quality assessments may be subject to error, proper contingencies for risks should be included.

As a simplified hypothetical example, let us suppose that only one quality attribute, namely mean-time-to-failure, has been specified as the total quality measure of a particular software product. Let us further suppose that C_t = \$60,000 represents the current expenditures for development, test, and repair that have resulted in a measured mean-time-to-failure of 4 hours, and that an additional ΔC = \$40,000 has

been forecasted as necessary to bring the system within its specified mean-time-to-failure (MTTF) goal of 24 hours. However, only \$20,000 remains in the budget, and this expenditure will only bring the MTTF to 12 hours. The current relative quality index is

$$q = \frac{60,000}{60,000 + 40,000} = 0.6.$$

In this case, 60% of the quality goal has already been reached. But when all funds will have been expended, the quality index will only have reached 80%, and yet the MTTF goal is a factor of two off! Either the original estimate of \$80,000 was off by 25%, or the project has been mismanaged with respect to quality concerns. The quality index itself does not reveal which of these possibilities is the case. It is a simple measure of quality status with respect to a plan.

But combined with a QBS, the q index time line can be much more revealing. We illustrate this utility by supposing that the project has decomposed its work in developing a system into a hierarchy of work elements for system design (including requirements, operational concepts, and system architecture), subsystem developments, and system integration, test, and delivery. We further suppose that the project has allocated the total estimated system cost ΔC across the w work packages and has devised accounting procedures that will track the total, current (time = t), and estimated runout work package costs, respectively, C_i, $C_{i,t}$, and ΔC_i, $i = 1,\ldots,w$.

The particular quality breakdown method and development process (e.g., waterfall, spiral, incremental delivery, rapid prototyping) are unimportant to the measurement of relative quality, as long as quality criteria have been assigned to each work element. The breakdown method *is* very important to being able to use q to manage the quality effectively, because the benefits in each quality attribute then come into play. Synthesis of the QBS from benefit considerations will be discussed further in section 7.

Since all work has been allocated to separate packages, the total costs, by definition, are *linearly* related to work package costs. We therefore find, after some mild algebraic manipulation, that the overall system relative quality is a weighted sum of the individual work package relative quality indices,

$$q = \sum_{i=1}^{w} c_i q_i . \tag{2}$$

The coefficients $c_i = C_i/C$, $i = 1,\ldots,w$ are the work package relative costs. The work package relative quality indices are given by

$$q_i = \frac{1}{1 + \varepsilon_i} \tag{3}$$

with $\varepsilon_i = \Delta C_i / C_{i,t}$.

If we aggregate a number of work packages together for management purposes, say as system design, subsystem development or integration efforts, or for quality reasons, such as packages to improve reliability or the user interface, then we similarly may define the relative quality index q_A of the aggregate A as the ratio of the current aggregate cost $C_{A,t}$ to the total required aggregate cost C_A, then we find that

$$q_A = \frac{1}{1+\varepsilon_A}, \tag{4}$$

$$c_A q_A = \sum_{i \in A} c_i q_i. \tag{5}$$

Here, $c_A = C_A/C$ is the relative cost of the aggregate, and $\varepsilon_A = \Delta C_A/C_{A,t}$ is the current quality deficit in aggregate work. If $A, B, \ldots, Z$ represent a set of non-overlapping aggregates that span the entire QBS, then

$$q = c_A q_A + c_B q_B + \ldots + c_Z q_Z. \tag{6}$$

Notice that the parameters q_i, q_A, ε_i, and ε_A at the work package and aggregate levels take forms similar to q and ε at the system level. The aggregate quality index formula in equation (6) is the more general form, as it reduces to equation (2) when the aggregate is the entire QBS, and to equation (3) when the aggregate is a single work unit.

We may carry the QBS work package decomposition to whatever level of detail is supported by project cost accounting procedures and limited by the effort required to devise and maintain it. We may also aggregate unit tasks to any degree deemed appropriate as milestones. Equation (3) provides a means for readily calculating the overall system and subsystem (or other aggregate) quality indices, regardless of how many layers of decomposition comprise the system or how quality attributes are allocated over the work packages in the aggregate. Equation (3) also provides a means for recognizing work aggregates displaying poor quality performance. For simplicity, we shall drop the distinction between work packages and work aggregates. The formulas are the same, except for interpretation of the subscripts.

If a work package or aggregate expends its resources before its q_i reaches unity, then quality goals have been compromised. Either further funding will be required, or the quality deficit must be made up by taxing another work package or aggregate, or else the loss in quality must be accepted by the customer.

The schedule required by a project is the sum of work package durations along a time-critical path dictated by resource precedences and quality requirements. Software work package costs typically derive only from work effort, measured in staff size times duration. In such cases, the schedule, too, can be expressed in terms of quality indices,

$$T_{crit} = C \sum_{critpath} \frac{q_i}{s_i r_i}, \tag{7}$$

where s_i is the number of full-time-equivalent staff and r_i is the average worker pay rate in the ith work package. (We note that q_i may depend on s_i, due to the team communication work factor, discussed in [Brooks 1974].)

7. QUALITY IMPROVEMENT STRATEGY

At each feasible opportunity, a project should examine its remaining resources $\tilde{C} - C$ and reallocate them toward reaching the promised quality goals, or, falling short of these, toward achieving the best customer satisfaction it can under the circumstances.

So, how should the quality attributes be serviced, and in what order? What is the best profile for expenditure of costs? The answers to these questions depend on the rates at which benefits accrue from various candidate cost expenditures. The overall quality index gives no direct assistance in deciding these questions. The increase in relative quality index at time t is independent of which particular work package expends its resources, for, as it turns out,

$$\frac{\partial q}{\partial C_{i,t}} = \frac{1}{C}, \tag{8}$$

which is a consequence of the fact that q itself only measures cost status. The partial derivative in management circles is called "leverage". The equation above says there is no cost leverage advantage in the quality index for any particular work package.

But perceived benefits do vary with distributions of costs, for users hold some quality attributes as being more important than others. In order for managers to act effectively, they must prioritize work towards the greatest user satisfaction to define the best work profile for the given customer expectations and budget constraints. This is where the choices of QBS decomposition method and development process become important.

The QBS and development process decisions are made through assessments of alternatives with respect to costs and benefits, a process that may take the form of informal analyses, analogies, rules-of-thumb, or formal evaluations using cost-estimation relationships or other forecasting methods. Today these decisions are largely made informally, based on experience, lessons learned, various management methodologies, and folklore, although some Operations Research formalisms have also been used where supported by good models.

The fundamental principle of Operations Research (OR) is that **if** one can create a model that accurately interrelates the characteristics of a product or set of products to the structure, procedure, and parameters of the process that developed them, and **if** one can quantitatively express the criteria by which to judge the quality of products, then it is possible to optimize the allocations of resources within the process. An OR approach to management thus depends on the "big **if**s" being satisfied to an acceptable degree.

We shall briefly outline the software quality management method and development of the QBS in terms of a formal OR model. Reflections of good informal practices may be seen echoed in the mathematical forms.

Further, we shall adopt vector notation to contain the information on quality indexes, costs, etc. For example, $\boldsymbol{q} = (q_1,\ldots,q_w)$; the vectors $\boldsymbol{\varepsilon}$, $\boldsymbol{c}$, $\boldsymbol{C}$ and $\boldsymbol{C}_t$ are similarly defined.

Formally or informally, there exists a benefit[3] function, here denoted by $B(\boldsymbol{q})$, that forecasts the current benefit in terms of the quality attributes. A formal expression of benefit forms a basis for repeatable operations-research analyses of alternative actions. Informal benefit assessments that are consistent with good management practices may be less repeatable, but are still effective.

Ordinarily, the benefit function relates to levels of quality achieved in the products to which it is applied. But time has a benefit value, as well, and timely delivery is important. We shall ignore this time dependency momentarily, but then return to such considerations later.

Let $\boldsymbol{\alpha} = (\alpha_1,\ldots,\alpha_n)$ denote the vector containing the current values of the n quality attributes α_i of interest, and let $\beta(\boldsymbol{\alpha})$ quantify the benefit as a function of each of these attributes.

We shall here adopt the convention that an increase in α_i yields an increase in the quality benefit. We may do this without loss, for if a proposed measure of an attribute were to lead to a decreasing benefit, then its negated or inverse value could be used instead. For example, if frequency of failure were proposed as a reliability quality metric, benefit would be judged to decrease if the frequency were to increase; but the inverse, mean-time-to-failure, when increased, would be judged to be beneficial.

Next, let $\boldsymbol{\alpha}(\boldsymbol{q})$ represent the quality-attribute versus quality index forecasting function (actually, an $n \times w$ matrix of functions). Then, the function $\beta(\boldsymbol{q})$ that relates benefit to current quality status is the composite form

$$B(\boldsymbol{q}) = \beta(\boldsymbol{\alpha}(\boldsymbol{q})). \tag{9}$$

This form of the benefit function separates trade-offs into two (almost) independent assessments: first, deciding how to quantify the composite benefits of multiple quality attributes; and second, deciding how work elements contribute to individual quality attributes. These assessments are not quite independent because the cost of a given level of quality almost always mitigates the perceived benefit of that quality.

The former assessment is primarily customer-related. TQM demands this be given high priority, as it directly relates to satisfaction of customer needs, without regard to cost. Formulation of the $\beta(\boldsymbol{\alpha})$ benefit function is apt to be subjectively derived, at least until industry has quantified, catalogued, and iteratively revised a few examples. We see informal examples today in the form of prioritized "wish lists".

[3] Also called the objective, utility, or value function in other contexts.

The second functional assessment is almost purely related to the software development process; TQM demands this be given high priority also, since higher quality products are deemed to spring from higher quality processes. Formulation of the $\boldsymbol{\alpha}(\boldsymbol{q})$ function is purely a matter of measuring and quantifying the repeatable quality aspects of a software process. This matrix of quality-cost relationships is the current quantitative limitation of the method of this paper, for many of the constituent elements have not yet been satisfactorily established formally. We may therefore proceed formally with the method mathematically, but only informally in the assessments of quality required in practice of the method.

Quality attributes above are expressed in terms of $\boldsymbol{q}$, rather than cost directly. This is done for normalization purposes. Since $\boldsymbol{q}$ measures the accrued costs relative to the total costs required for quality (as opposed to costs relative to the budget), the required quality attribute goals are contained in the vector $\boldsymbol{\alpha}(1, 1,\ldots,1)$. If a given attribute quality were expressed as $\alpha_i = f(\ldots, C_i, C_{i,t},\ldots)$, we could equally well express it as $\alpha_i = g(\ldots, C_i, q_i,\ldots)$ by substituting $q_i C_i$ for $C_{i,t}$ in $f(\ldots)$.

The operations research problem is thus to find the particular vector $\tilde{\boldsymbol{q}}$ that produces maximum end-of-project benefit under the budget constraint $q_{final} = \tilde{q}$. The approach is classic optimization of $B(\boldsymbol{q})$ over values of $\boldsymbol{q}$ whose coordinates are constrained and described by

$$\sum_{i=1}^{w} c_i q_i = \tilde{q} \quad \text{for } q_i \geq 0, \quad i = 1,\ldots,w \tag{10}$$

and there is a further constraint imposed by equation (7). Together, these restrict the number of degrees of freedom in $B(\boldsymbol{q}_{final})$ for optimization.

An extremum can either be global (truly the best value) or local (the best in a finite region). Virtually nothing is known about finding global extrema in general. Many optimization approaches are embodied as computer algorithms available in subroutine libraries. Classical considerations are computer speed, memory, and computational cost. Approaches to finding extrema resort to trial-and-error strategies, a number of which are described in [Press 1987]. Many of the algorithms rely on the function gradient (multidimensional slope function) $\nabla B(\boldsymbol{q})$ to point in the direction of increased benefit.

Incremental steps in quality $\delta\boldsymbol{q}$ over time result in incremental changes in benefit δB, which we desire to reach the final optimum values indicated above. The time rate of quality benefit gain is

$$\dot{B} = \phi_1 \frac{dq_1}{dt} + \ldots + \phi_w \frac{dq_n}{dt} = \boldsymbol{\phi} \cdot \dot{\boldsymbol{q}} = \nabla B \cdot \dot{\boldsymbol{q}}, \tag{11}$$

where $\dot{x} = dx/dt$ represents time rate of change. The gradient coefficients ϕ_i are given by

$$\phi_i = \sum_{j=1}^{n} \frac{\partial \beta}{\partial \alpha_j} \frac{\partial \alpha_j}{\partial q_i}. \tag{12}$$

Each ϕ_i coefficient measures the rate that benefit derives from an expenditure in q_i. Each term $\partial\beta/\partial\alpha_j$ expresses the change in benefit with respect to quality attribute α_j, and $\partial\alpha_j/\partial q_i$ relates the effect of an incremental expenditure in work package i to the improvement of quality attribute α_j. Some terms may be negative when an expenditure towards servicing one quality attribute has a counter-impact on another.

The gradient $\boldsymbol{\phi}$ indicates the local *preferred* direction in quality improvement at each point in time. As each quality attribute α_j reaches its goal value, further improvement of α_j is not required, so $\partial\beta/\partial\alpha_j$ becomes 0, and no longer contributes to ϕ_i.

When a gradient term ϕ_i is positive, an expenditure in work package i is beneficial. If the term is zero or negative, then no expenditure should be made (unfortunately, we cannot unspend money!). If all the terms were equal and positive, the application of δq toward improving any one of the attributes would, in judgment, yield the same incremental benefit. Having one ϕ_i larger than the others would indicate that higher immediate benefit per unit of cost would come from work package i.

For a given size quality increment $\delta q = c_1 \delta q_1 + \ldots + c_w \delta q_w$, each individual δq_i may be chosen so as to maximize the total overall incremental benefit. The classical approach would demand that the entire δq be spent on that work unit having the greatest ϕ_i (this is a consequence of the linear constraints of equations (7) and (10) above). This approach would lead to expending the project's resources serially, executing work packages in order of judged benefit.

However, as a practical matter, time-benefits require that critical path work must proceed in the order scheduled. Work off the critical path is more flexible and takes place in parallel. The assessment of incremental benefits versus incremental quality indices, nevertheless, serves to reinforce the importance of critical path items. If some work on the critical path is deemed of less benefit than some work off the path, then some rethinking of what constitutes benefit may be in order.

8. WHAT PROJECTS CAN DO NOW

Formal methods may be applied toward optimizing quality when project benefit functions eventually become quantified. But even if unquantified, the benefit functions do exist in informal guise, and managers do use these, perhaps unconsciously, when making decisions. We can look to the formal approaches for guiding principles when allocating expenditures under informal circumstances.

Good managers have applied mathematically sound guidelines for years, perhaps even without knowing it. Using their experience, knowledge base, and pertinent project information, including predefined priorities for quality, they develop budget profiles that optimize expenditures in accordance with their perceived requirements

and constraints. They assess progress relative to the expenditure profile, making readjustments as the priorities or circumstances change. By taking a more global view of quality than just performance and reliability, they are able to achieve better overall products. By recording their priorities, work breakdowns, and quality status metrics over project lifetimes, they bequeath to future projects and quality modelers the means for improvement.

We wish now to show how these informal principles are consistent with formal optimization processes. We shall illustrate the equivalence using a "follow-the-gradient" approach. We choose this approach not because it is the fastest for a computer (in fact, it is usually not), but because it is simple in form, intuitive, and requires only shorter-range forecasts of benefit versus resources.

Properly done, the planning of work packages should align priorities with the gradient of the benefit function, resulting in weights for the way costs are applied within each increment of the project. That is, expenditures over each δt schedule increment would produce a change in the quality index of δq whose individual components δq should be in the same direction as the gradient ϕ, or

$$\dot{q} = h\phi. \tag{13}$$

By equation (12), assessments of priority should examine the combined effects of how much each quality attribute incrementally contributes to benefits and how much the work in each package contributes to the quality attributes. Using equation (2) permits us to eliminate h in the above equation and to express the prioritized quality allocation condition as

$$\dot{q} = \left(\frac{\dot{q}}{\sum_{i=1}^{w} c_i \phi_i} \right) \phi. \tag{14}$$

The gradient-directed strategy is thus to set priority factors for each of the project schedule increments in the direction of the benefit gradient, to allocate expenditures accordingly, and to repeat this process during the development process whenever status demands.

This incremental heuristic strategy does not always produce a result globally optimized over all ways of running a project, but it applies well when the particulars of a global optimum are unknown. Recording lessons-learned for posterity permits other projects perhaps to recognize more globally optimized strategies for continual improvement.

A more formalized statement of the incremental priority-driven quality management process is merely a restatement of good management practice:

(1) At each decision point in time, estimate the cost to complete and the current quality status, and compare these with available resources and quality requirements.

Then set priorities for expending the remaining resources, considering the contributions of each particular quality attribute to benefit, and of each work unit to quality attribute improvement. Use whatever methods, metrics, and statistics are at hand for assistance.

(2) Allocate incremental resources in proportion to assessed priority, and then extrapolate the probable status of the project at the next decision date.

(3) Continue to apply these "what if" estimations at simulated next-decision opportunities, until the required quality is perceived to have been reached, or until the budget will have been exhausted.

(4) If the current plan falls short, reassess the planned expenditures using other project alternatives that may exist, and reapply these steps until the best solution seems to have been found.

(5) Allocate resources for the next stages of development according to the best plan known.

(6) Record priorities, allocations, cost drivers, and perceived quality status for future insight, lessons learned, and modeling purposes.

9. CONCLUSION

This paper has introduced the concept that relative quality quantification, measured in terms of amounts of work performed and required, together with a quality-oriented WBS and means for assessing the benefits of each work package, form a potentially valuable method for unifying quality requirements and measurements through the same managed units for all quality attributes. The concept appears to apply throughout the hierarchic layering of system and subsystem architectures.

Quality-attribute-versus-cost trades, when they can be formulated, enable the impacts of given quality requirements to be evaluated in terms meaningful both to management and to customers. Comparison of actual and predicted quality status may serve as a useful management indicator, and may also provide feedback data for improving quality assessment methods earlier used. Relative quality index tracking may contribute to a quantitative strategy for improving cost-effectiveness through prioritization of work and assessment of incremental benefits.

Most importantly, this paper suggests that a more comprehensive and useful quality measure may also exist. The relative quality index concept appears to be extensible to other critical-resource-related parameters other than dollars, such as workyears, schedule, facilities, and capacity.

The recognition and adoption of these principles by software professionals will, it is hoped, provide a clearer direction and unifying focus for future TQM management research.

REFERENCES

Belady, L.A. (1977), "Software Complexity", Software Life Cycle Management Workshop, AIRMICS, Atlanta, GA, pp. 371–383.

Boehm, B.W. *et al.* (1978), *Characteristics of Software Quality*, Elsevier-North Holland Book Co., Amsterdam, Holland.

Boehm, B.W. (1981), *Software Economics*, Prentice–Hall, Inc., Englewood Cliffs, NJ, pp. 310–313.

Brooks, F.P. (1974), "The Mythical Man-Month", *Datamation 20*, 12, 45–52.

Crosby, P.B. (1979), *Quality is Free*, Mentor Books, New York, NY.

Curtis, W. (1981), "The Measurement of Software Quality and Complexity", In *Software Metrics*, A.J. Perlis *et al.*, Eds., MIT Press, Cambridge, MA, chapter 12.

Deming, E. (1989), *Out of Crisis*, MIT Center for Advanced Engineering Study, Cambridge, MA.

Elsayed, E.A., G. Taguchi, and T. Tsiang (1988), *Quality Engineering in Production Systems*, McGraw–Hill Book Co., NY.

Gilb, T. (1977), *Software Metrics*, Winthrop Press, Cambridge, MA.

Humphry, W. (1987), "Characterizing the Software Process—A Maturity Framework", Technical Report CMU/SEI-87-TR-11, ESD-TR-87-112, Software Engineering Institute, Carnegie-Mellon University, Pittsburg, PA.

"Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for Their Use", International Organization for Standardization, ISO/TEC 9126.

Koontz, O'Donnell, and Weihrich (1984), *Management*, McGraw-Hill Book Co., NY, Eighth Edition.

Lyu, M.R. (1991), "Measuring Reliability of Embedded Software: An Empirical Study with JPL Project Data", *International Conference on Probabilistic Safety Assessment and Management*, Beverly Hills, CA, pp. 493–500.

Mosemann, L.K. (1993), "Creating a National Vision and Force in Software Through Software Measurement", keynote address, Cooperstown I Workshop, Rome Laboratory, Griffiss AFB, NY.

Pall, G.A. (1987), *Quality Process Management*, Prentice–Hall, Inc., Englewood Cliffs, NJ.

Perlis, A.J. *et al.* (1981), *Software Metrics*, MIT Press, Cambridge, MA.

Phadke, M.S. (1989), *Quality Engineering Using Robust Design*, Prentice–Hall, Inc.

Press, W.H., Ed. (1987), *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, New York, Chapter 10.

Stamm, S.L. (1981), "Assuring Quality Quality Assurance", *Datamation 27*, 195–200.

Tausworthe, R.C. (1980), "The Work Breakdown Structure in Software Project Management", *Journal of Systems and Software 1*, 181–186.

Thayer, R.H. (1983), "Software Engineering Project Management: A Top-Down View", *Software Engineering Project Management*, IEEE Computer Society Tutorial 751, Computer Society Press, Washinton, DC.

Walton, M. (1988), *The Deming Method*, Dodd-Mead, Inc., New York, NY.