From the Publisher

# Factoring Process Improvement into the Awarding of Sustainment Contracts

Lt. Col. Joe Jarzombek
*ESIP Director*

Reductions in procurement funds mean that most of the U.S. military's year 2010 systems are already in our current inventory. Because modification of software-intensive systems provides the most promise for increases in system capability and flexibility, many 2010 requirements will be achieved through sustainment of existing systems. With so much being dependent upon the successful upgrade of existing systems, perhaps it is time to assess how process improvement efforts might be better factored into "best value" comparisons associated with the award of sustainment contracts.

Process improvement requires an investment of time and resources, which in turn raises direct labor and overhead costs. Development organizations make this investment because their increased efficiency and quality translate into higher profitability and more follow-on contracts. Because such organizations usually produce systems in less time and with fewer defects thus lowering development costs, process improvement can be factored into bids associated with new system deliveries.

On the other hand, sustainment contracts normally involve "level of effort" tasks and are negotiated based on labor rates for defined periods and funding levels. Therefore, labor rates weigh heavily in the determination of best value, and unfortunately, process improvement efforts are often difficult to quantify relative to labor rates. Indeed, an organization that uses low-skill-level employees and invests little in process can offer low labor rates. However, studies demonstrate that those same organizations take longer to deliver capabilities that have more post-deployment defects. Contracts awarded to low-labor-rate organizations can easily result in higher total costs and inferior results.

Most source selection teams understand that process improvement contributes to "best value"; yet they also know today's "protest prolific" contracting environment makes it difficult to award sustainment contracts to higher-labor-rate organizations—even those likely to provide the best value—without quantifiably objective criteria such as industry standards. This has fundamentally dire consequences for the military's 2010 capability unless sustainment contracting policies and practices accommodate provisions for process improvement.

Integrated capability maturity models (as opposed to single discipline models) provide the best process improvement guidance for organizations that provide post-deployment support. For fielded systems, sustainment includes additional acquisition, development, modification, and maintenance activities, cutting across disciplines that are often compartmentalized within different departments. Therefore, enterprise-wide process improvement is critical to sustainment organizations. That is why the Capability Maturity Model Integration (CMMI) effort will better support the institutionalization of enterprise-wide process improvement (see CMMI at http://www.sei.cmu.edu). The Federal Aviation Administration (FAA) has already demonstrated the value of using an integrated CMM (iCMM) with staging guidelines (see FAA-iCMM® Web site and "Smart Buying with the Federal Aviation Administration's Integrated Capability Maturity Model" on page 15 of this issue).

Perhaps integrated process improvement efforts might help support the creation of a labor rate standard that gives higher-maturity organizations due credit for their higher efficiency. This would require documentation of the increased productivity of organizations with higher maturity ratings. Many organizations use the industrial engineering "standard hour" of work to estimate and price a level of effort. We need a method to quantify what a "standard software engineer" can produce in one hour in a "defined capability and maturity environment." If this could be determined, the software industry might be able to tie a "productivity compensation factor" to the organization's maturity to equalize unfair bidding advantages between competing organizations of different maturity levels. For example, the standard could authorize CMM Level 1 organizations to budget efficiency at 95 percent, Level 2 at 100 percent, and through to Level 5 at 150 percent. Some could argue that these numbers are not even close to the increase in productivity; however, it shows the need to invest in discovering what the real numbers are.

More widespread recognition is needed to substantiate that overhead associated with process improvement, while it increases labor rates, reduces the cost of sustainment. Merely awarding sustainment contracts based on lowest labor rates could have irreparable consequences for our 2010 capabilities. ◆

Letter to the Editor

## Summarizing Meetings Is Vital

*CROSSTALK* is beneficial, and I endorse it every chance I get. I really enjoyed "Rules a Program Manager Can Live By," *July 1998*. I especially identified with Step 7, "Summarize Meetings." All too often, people get burned by not doing this. Everyone seems to come away with something different. I have been emphasizing the importance of summarizing meetings, and quoting your article.

Al Kaniss
*U.S. Navy*
*Patuxent River, Md.*

# Driving Quality Through Parametrics

Daniel D. Galorath, Lee Fischman, and Karen McRitchie
*Galorath Incorporated, The SEER Product Developers*

*In this article, we show how prediction models are used to improve delivered quality. We further show that if you can anticipate and plan for the factors that affect quality, you can leverage quality management activities to improve the entire development effort.*

Software development models[1] are gaining acceptance in the software project estimating community, which is always challenged to establish cost and schedule objectives before projects begin. Predictive models can in fact be further deployed into software projects to improve the quality of development.

Developers implicitly understand the notion of software quality; however, many ideas about quality unfortunately go no farther than active prevention such as testing or walk-throughs.

As critical as active quality control is, good planning will multiply the effectiveness of any effort, saving both time and money. But how can plans be laid without fully anticipating the factors affecting quality? In the haze of battle surrounding most development efforts, software development models provide the answer.

## What Is Quality?

Discussions about software quality all too often focus on a single measure: defects delivered. Indeed, this may be the most significant measure of quality because software is useless—or worse—if it suffers from too many bugs. However, as with any other product, there are many dimensions to software quality.

- Correctness. Is the program correctly specified?
- Usability. Can users learn to use the software with reasonable effort?
- Efficiency. Does the software minimize the use of hardware resources?
- Reliability. Is the mean time between failures sufficiently long?
- Adaptability. Can the software be easily adapted to new uses?

Table 1. *Cost/schedule/defect trade-off report.*

| | More Testing | Less Testing | Difference |
|---|---|---|---|
| Development Schedule Months | 31.54 | 29.89 | 6% |
| Development Effort Months | 1,422.39 | 1,210.15 | 18% |
| Development Base Year Cost | $20,909,062.00 | $17,789,243.00 | 18% |
| Defect Prediction | 33 | 60 | -45% |
| Constraints | MIN TIME | MIN TIME | |
| Peak Staff | 63.59 | 57.10 | 11% |

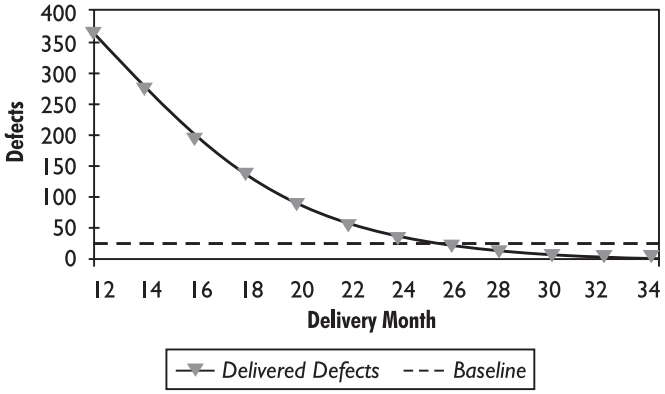



Figure 1. *Defects delivered vs. length of development.*

- Robustness. Can the software be stressed without breaking? Does it stand up to intentional or negligent user abuse?
- Maintainability. Once delivered, how challenging is it to maintain the software?

Software development models can directly account for many of these quality factors, either directly through estimates, i.e., defects delivered, or via parameter settings that in turn drive estimates.

## An Overview of Parametric Models

Parametric models allow developers to specify software project variables and to receive in return estimates of effort (cost), schedule, and defects. Variables typically include complexity of the software to be developed, specification and test level, quality of the development staff and tools, complexity of the development language, and software size. Vendors of more mature tools have had a longer opportunity to collect data and perform enhancements, so more variables are generally available for their models.

Parametric models have several advantages over other methods of prediction. First, vendors work continuously to assure that their tools are accurate. The better tools also can be substantially calibrated to the specifics of an organization while retaining the essential sensitivities of key parameters. These tools give rapid, elaborate feedback and therefore can be used for realistic trade studies, even in a collaborative mode with "heads up" conferencing features.

For the concurrent engineering necessary to simultaneously satisfy cost, schedule, requirements, and quality goals, the benefit of parametrics is clear. No other method permits such rapid, elaborate interaction between varied

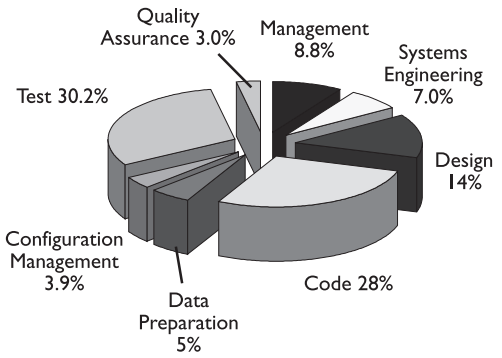



Figure 2. *Activity allocation chart.*

interests. Once development goals are set, parametric estimates can be used by developers to ensure that quality goals are achieved at least cost.

## Defect Prediction

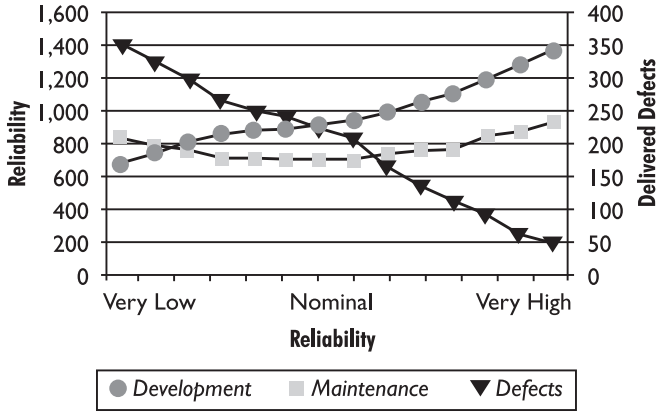
Of the many different aspects of quality, delivered defects are among the most obvious and quantifiable. A number of defect prediction methods are in use that rely on gross volume and complexity metrics such as size,[2] Halstead Software Science Volume, McCabe Cyclomatic Complexity, or other composite measures.[3] An "integrated" defect model allows defect predictions to be evaluated alongside of staffing and cost considerations, which opens up a world of comparative scenarios.

The most useful prediction for a quality model is *delivered defects*, meaning those that escape detection and are delivered to the end user. A defect prediction allows you to plan for acceptable magnitudes and take corrective actions—follow a better process, lay on further testing, reduce scope, lengthen schedule—when predictions are too high.

Table 1 illustrates a trade study driven by defect predictions. For the testing effort required to halve delivered defects, costs will rise somewhat, and schedule less so.

An alternative to predicting delivered defects is modeling *potential defects*. Doing this allows developers to engage in explicit defect-related "what-if" scenarios, such as illustrated in Figure 1. The dashed line shows the defect level given an optimal schedule that minimizes development effort. If the product is delivered earlier, defects will rise above this level, but if they are delivered later, defects will be lower. Note how the defect penalty decreases as the development schedule stretches. This is a powerful tool for managers to have to plan schedules to specific defect targets.

Figure 3. *Development and maintenance effort as quality assurance increases.*




## Modeling Promotes Active Quality Control

It makes sense not only to predict defect levels but also to predict adequate levels of quality control. Parametric models offer developers the advantage of a database of completed projects and industry wisdom. They show in concrete terms exactly how many employees are required to deliver a product of certain quality.

Models provide insight into quality control activities by parsing effort and staffing estimates into individual labor categories and activities. These parsing factors can often be adjusted to the organization and the development process. This not only makes staffing estimates more suitable to specific development environments but also allows organizations to better promulgate desired levels of quality control.

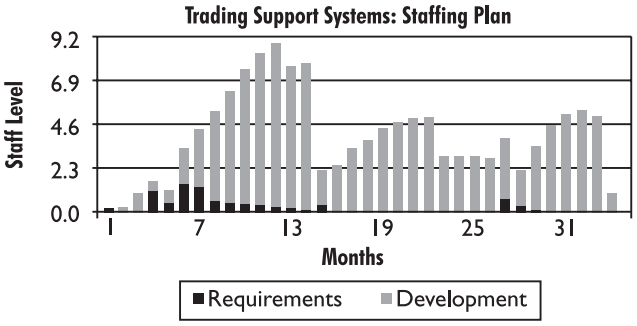



Figure 4. *Staffing profile for a large project.*

Is it necessary to rigorously follow the testing level indicated? The answer is that they are benchmarks only, indications of what past development teams have required in order to achieve defect efficiencies along the lines of those envisioned. Maybe, quality targets can be achieved with different test staffing from that indicated; maybe, they cannot.

## Accounting for Other Quality Factors Through Specification

The relationship in modeling is "many to few"—many parameters are available in a parametric model to specify factors from the development environment to the final product. The predictions that result are usually limited to the trinity of cost, staffing, and defects.

This mix of estimates and parameters allows developers to account for many quality factors, as indicated by estimates or as specified by parameters. With parameters, the analyst is not *predicting* quality, as in the case of defects, but rather is *specifying* quality, then judging the impact on cost, schedule, and defects. Aspects of quality handled via parameters include efficiency, adaptability, robustness, and maintainability.

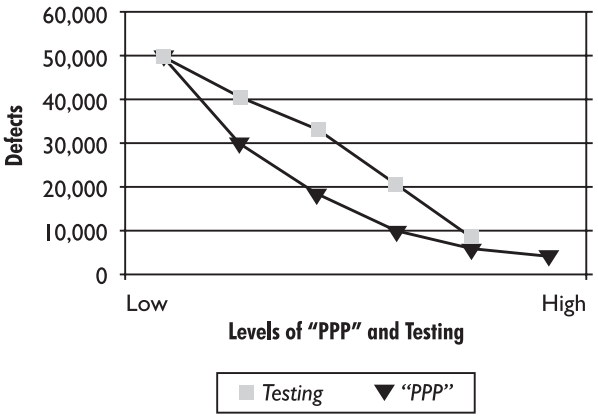



Figure 5. *Impact on defects: team quality vs. testing.*

As an example, maintainability is strongly correlated with maintenance costs, which can be modeled by varying other parameters. Figure 3 shows how reliability levels, for example, impact not only upfront development costs but also eventual maintenance costs. As specified reliability increases, defects decline, but development costs increase. Maintenance costs may also decline with increasing reliability, but soon the specification for a system becomes so rigorous that maintenance costs also rise.

Some quality-related attributes are not specified with parameters but can be accounted for in other ways. For example, usability may impact size, which is a prime input to a development model. Cost and schedule estimates will vary in direct proportion to software size, matching the intuitive result that greater quality comes only at a higher price. Development models tell you exactly how high that price will be.

## Improving Quality by Modeling the Development Process

The sheer utility of development models' planning features offers other avenues toward improving quality. With insight into your project, as Figure 4 suggests, you are much closer to engineering quality into your development process.

All too often, quality slips when staffing requirements are poorly anticipated. Knowing the optimal staffing profile for a project improves planning, lessens staffing-related volatility, and therefore permits the timely application of testing activities. This particular chart also makes explicit the proper mix between early requirements work vs. coding and testing; as is well known, sufficient requirements definition does more to determine quality than testing.

Development models simulate reality by incorporating known development dynamics. For instance, it is less expensive to do good work first than to apply more stringent testing later. Figure 5 illustrates such a trade-off between the three P's of "people, process, product" vs. the alternative of increased testing. The impact on defects delivered is shown; either curve is drawn holding the other factor constant. Notice how there is a disproportionate return on improvements in the development team, whereas there is only a linear return on improvements in testing.

## Conclusion

Although parametric models have long been used to establish cost targets, they can be used for much more. Modern software development models have years of analysis support invested in them so that they can address such dynamic management issues as quality. If your organization uses parametric modeling for its estimates, see your estimators to see what they and their tools can do for you. ◆

### About the Authors

**Daniel D. Galorath** is president of Galorath Incorporated. He has worked in all aspects of software development and software management and is one of the principal developers of the SEER-SEM software evaluation model. His published works include software cost modeling, testing, error prediction and reduction, and systems requirements definition.

**Lee Fischman** is special projects manager at Galorath Incorporated. He is a frequent consultant on estimating projects, and he also conducts core research and development of SEER software tools. He wrote the Software Evaluation Guide for the Office of the Secretary of Defence, Program Analysis and Evaluation, and he has explored software economics and estimating in numerous papers over the past several years, all available at http://www.galorath.com.

**Karen McRitchie** is vice president of development at Galorath Incorporated, responsible for design and development of SEER tools. She has nearly 10 years experience in software and hardware cost-estimating and hardware reliability modeling. She has been a lead member of several Air Force cost-estimating teams for major Department of Defense programs and has taught dozens of estimation methodology courses for costing professionals.

Galorath Incorporated, The SEER Product Developers
Voice: 310-414-3222
E-mail: support@galorath.com

### Notes

1. Mathematical estimation models are known to the cost estimating community as "parametric models." As understood in mathematical English, this implies that functional forms are prespecified. However, to costing personnel, parametric means only that these models have parameters to modify; no comment is being made about functional form.
2. Lines of code, function points, and object-based metrics are the most commonly used size measures.
3. For a description of how SEER-SEM handles defect prediction, refer to the "SEER-SEM Defect Prediction" technical note available at http://www.galorath.com.

# 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.

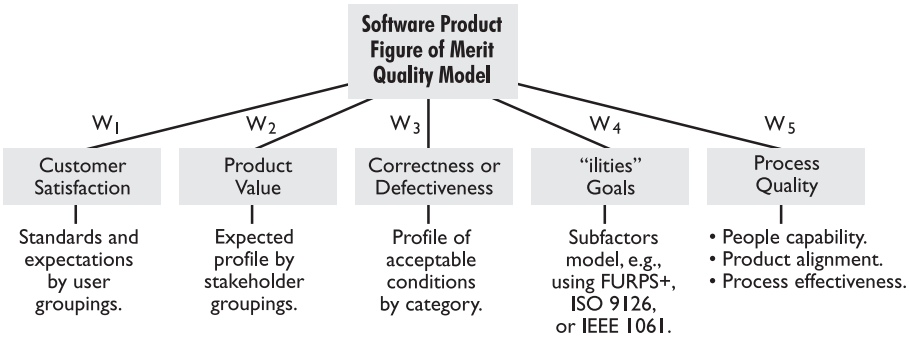
The categories and subcategories of the FOM model can be weighted as needed for use by domain or system. The FOM would be the sum of the weighted factors.

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.

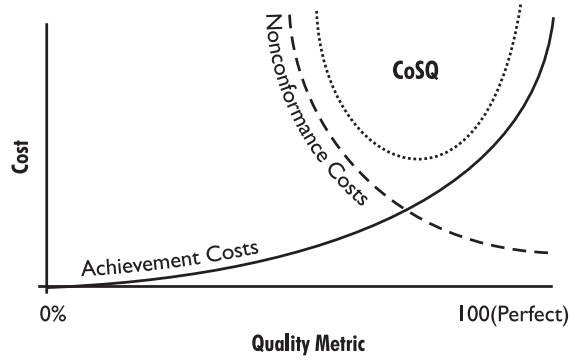
As an industry, we have collected little data about CoSQ that could be used either to challenge or validate this theory. The little we do have suggests that this economic model may not account properly for the dynamics of software development, since perfection is either not a goal or is a quickly moving 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], and the Raytheon efforts [11], which all discuss trends in software rework costs.

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



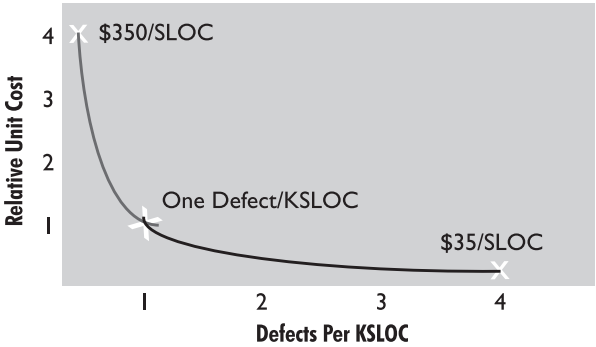




Figure 3. *The cost of ultra high reliability in the shuttle software.*

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.

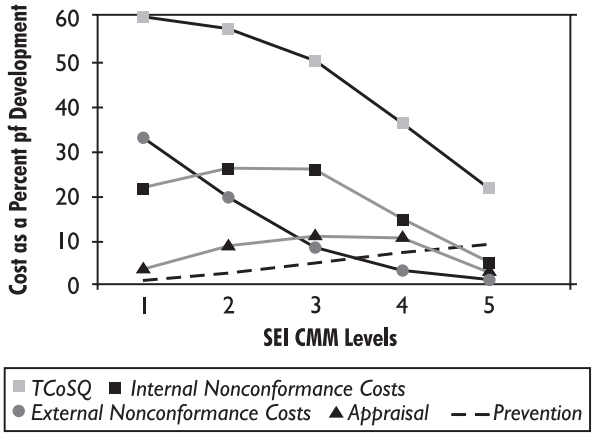
### Initiating the CoSQ Effort

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.

# The Software Quality Certification Triangle

Jeffrey Voas
*Reliable Software Technologies*

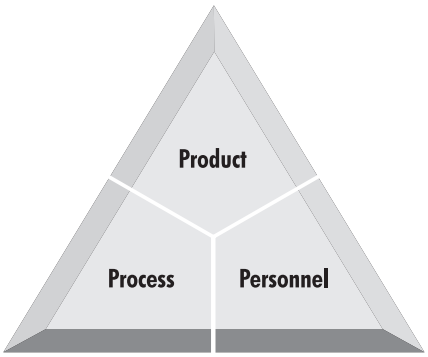
*There are three distinct approaches to certifying the quality of software: accrediting personnel, certifying the development organization, and assessing the "goodness" of the software. These approaches, and hybrids thereof, are described, and criteria are given to determine which approach is best, depending on the software that needs to be certified.*

Developing quality software is often considered elusive—it is more difficult to confidently know that you have developed good software than it is to build good software. In the physical sciences, the reverse is true—it is easier to measure the degree of perfection than it is to achieve perfection.

One reason why it is difficult to measure software quality stems from the many practical and theoretical deficiencies of software testing. For example, consider that to be 99 percent confident that a program has a probability of failure of less than one in 1 million, the software must be tested over 5 million times without observing a failure. Testing 5 million times requires that you have an oracle that is correct (an oracle is a person who knows or a program that knows what the correct software output is for all of the 5 million test cases). Rarely does a perfect oracle exist, and to create 5 million test cases would be intractable. And if you have the oracle and the test cases, there remains the impossible task of having to test using them.

Challenges such as these have made many in the software community decide that quality assessment of a software product is impractical. In addition to the traditional approach of assessing the "goodness" of the software, this has led to alternate approaches to software quality assessment. The two key competing approaches are process maturity assessment and accreditation of software professionals. The remainder of this article describes the pros and cons of these three approaches to predicting the quality of software.

Figure 1. *The software quality certification triangle.*




## Accrediting Personnel

There are various ways to accredit, i.e., certify, personnel. The rigor with which personnel are certified depends on the criticality of the services that the person offers.

Professional licensing examinations, practical experience, and earned degrees are a few ways in which professionals can be accredited. For example, graduating from law school says something about a person's ability to practice law. It says less, however, than had the person also passed the bar. If this were not true, there would be no need for state bar examinations.

The intuition behind certifying "people skills" is simple; it should not be left up to the untrained consumer to be responsible to determine whether a candidate is qualified to perform the desired services. For example, how can Joe Public be expected to determine whether a dentist is qualified? Only if Joe Public were a dentist would he have any hope of making such a determination. By requiring dental school graduates to pass an examination prepared by dentists, the state takes the responsibility away from Joe Public. Further, if certified professionals do not live up to the expectations of their peers, they could be found liable and could lose their certification.

Like the older and more traditional professions of accounting, medicine, and law, the software industry is beginning to standardize the core principles each software professional should know. Microsoft claims that there are greater than 160,000 people who have become Microsoft certified as either product specialists, solution developers, trainers, or systems engineers [1]. This type of certification is "voluntary" (not required by any official governing organization) and expensive; however, the costs of certification can be recouped in the first year of working from the extra income the certificate enables. For example, it costs from $8,000 to $12,000 to become a Microsoft certified systems engineer (MCSE), and the total time to certify is approximately six months [1]. A person then can expect to make the same amount in additional income compared to a person who is not MCSE certified.

Just like doctors, lawyers, and certified public accountants, rumblings are also being heard concerning *mandatory* software engineering personnel certification. A vote by the Texas Board of Professional Engineers on Feb. 18, 1998 stated the board's intention to recognize software engineering as a legitimate engineering discipline and stated plans to license professional engineers in software engineering (a complete position statement from the Texas board can be found at http://www.main.org/peboard/softweng.htm). On June 17, 1998, the Texas board gave unanimous approval to all proposals in the statement. Beginning July 1999, the Texas board will license software engineers who can satisfy the following [2]:

- Possession of an engineering degree, a computer science degree, or some other high-level mathematics or

science degree that the board will evaluate for adequacy.

- At least 16 years of creditable experience performing engineering work (12 years for those who hold a degree approved by the Engineering Accreditation Commission of the Accreditation Board for Engineering Technology, Inc.).
- References from at least nine people, five of whom must be licensed engineers.
- Submission of documented credentials as required.

After the Texas board releases the professional software engineering examination in 1999, individuals with less experience will be allowed to obtain a Texas Professional Engineering license by passing the examination.

## Assessing the Software Product

Generally, there are two approaches to product-based assessment of quality: white-box and black-box. White-box assessment techniques include activities such as collecting static code metrics or measuring the degree of coverage achieved during unit testing. Black-box techniques include reliability testing.

White-box and black-box techniques are not panaceas, however. For example, because reliability is based on logical correctness and the operational environment and not structural properties, it is unclear what relationship a code complexity metric has with the reliability of the software. Further, it is impossible to exhaustively test a simple program that reads in two 32-bit integers [4].

With today's push toward commercial-off-the-shelf (COTS) software, white-box certification techniques are normally not used by COTS consumers. However, white-box techniques may be applied by vendors if they wish to do so. Therefore, COTS consumers who are genuinely concerned about what lurks in the software they purchase must decompile back to source code to apply white-box analyses such as coverage testing or inspections.

Most COTS licenses deem this act a violation of the licensing agreement. Further, pending global legislation may weaken the ability of consumers to have such analysis done by independent corporations or consultants. In addition, a global treaty has been presented for U.S. approval entitled the World Intellectual Property Organization (WIPO) Treaty. The treaty includes language that makes it illegal to reverse engineer software to expose security vulnerabilities. The treaty will make it illegal for corporations and consulting services to conduct real-world testing of security software. Supposedly, research organizations will still be allowed to do so, however.

President Clinton has announced his intentions to sign the treaty, and it is expected to pass in the U.S. House of Representatives. The U.S. Senate has already passed the measure that deals with the treaty by a score of 99 to zero. The legislation is part of a global attempt to produce treaties that reduce the amount of copyright infringement on information technology. But the downside is that it disallows consumers the right to independently certify the security of the software they purchase (without the vendor's permission).

## Certifying Processes

Because of the limitations associated with different forms of product assessment (testing as well as techniques such as formal verification), in the mid-1980s, the notion of "directly assessing software quality" was dismissed as implausible. This opened the door to ideas such as "process maturity assessment" and other indirect approaches. The most well-known process assessment model is the Software Engineering Institute Capability Maturity Model[SM] for software. This model and other manufacturing-like standards rely on one premise—good processes deliver good software. This premise has also lead to government regulatory standards for software certification in avionics, medical devices, and electric power generation. The premise here is plausible. All developers have to do is score themselves using a pre-defined ranking scheme (for what is and is not good software development procedures), then apply that score to their software. For example, if development organization A is ranked higher than organization B, it is assumed that software from A has more quality than software from B. The problem is that good processes do not guarantee good software [6]. If performed properly, good processes merely increase the likelihood of producing quality products; if processes are not performed properly, the likelihood is reduced. However, given a fixed set of development processes, it is still possible that organization A, that improperly applies the set, produces better software than organization B, that properly applies the set. Furthermore, this does not account for issues related to which processes are "best." These facts, taken together, diminish the notion that process assessment will become a satisfactory substitute for product assessment. Ask yourself this: Would you buy a car without test driving it? Few would, but this is precisely what is done when process assessments are employed instead of product assessment. Process assessments are analogous to a car manufacturer that tells you what phases were undertaken during manufacture, which is no substitute for taking a test drive.

## Software "Insurability"

I wil examine what role quality certification can play with respect to software insurability. Software insurability refers to the software-induced risk that an insurer is willing to take in exchange for an insurance premium. The insurer is not insuring the software but is instead insuring the object that the software controls. But before offering insurance for that object, the insurer must understand the worst-case scenarios that can result if the software is defective.

Consider that Swedish insurer Trugg-Hansa made the following exclusion effective May 1, 1998 in the general conditions of its business insurance policies.

> "The policy will not cover damage, cost, legal, or other liability caused directly or indirectly or connected to time-related disturbance in computer functionality."

This demonstrates the extreme, defensive posturing being seen as a result of the year 2000 problem. But of equal significance, it opens the door for

nontime-related exclusions for other anomalous software behaviors. For example, exclusions might someday read as follows:

> "The policy will not cover damage, cost, legal, or other liability caused directly or indirectly or connected to disturbances in computer functionality."

Such a waiver enables an insurer to avoid responsibility for all computer-related problems. The onus is placed on consumers to know the quality of the computer systems they employ. Consumers now bear their own liability without access to an insurer to step in as their surrogate in case of a mishap. This represents a first in the software industry—insurers are so concerned about software failures that they have begun to include exclusions in their policies. When a situation such as this is coupled with the WIPO Treaty and the disregard for consumer protection that exists in the current version of the Uniform Commercial Code, Article 2B [3, 5], it is clear that the need for independent third-party certification concerning the processes, product, and personnel could not be greater.

Interestingly enough, a business has been formed to address the insurability problem—the Software Testing Assurance Corporation of Stamford, Conn. This company was founded in 1997 to provide independent certification. Their first certification offering will assess the testing processes used on year-2000-converted software. They currently certify most process assessments and a small portion of product assessments (their standard can be viewed at http://www.STACorp.com/draft/standard.htm). This independent certification is available only to corporations that seek business disruption insurance in the event their computer systems fail as a result of year 2000 software problems. The founding of this organization opens the door for additional software quality certification standards for information systems when business risks are directly tied to software quality and insurance protection is sought.

## Summary

The hypothesis that certified personnel equates to higher quality software is easy to disprove. The hypothesis that a more mature process equates to higher quality software can also be easily debunked. Product assessment that studies the dynamic behavior of software is clearly the best approach to certifying software quality, but problems that relate to feasibility often reduce the ability to perform assessments with any degree or thoroughness.

The best approach is to create a variety of different certification schemes based on the different types of examinations or processes used from each of the three categories and the criticality of the software (flight control software vs. games). That is, aspects of each of these three broad approaches can be combined into a single standard. For example, knowing that an organization has a certain process maturity, the personnel who developed and tested the software were licensed, and the software received certain forms of quality assessment should result in greater confidence in the software's quality than if only one of these facts were known. The challenge, naturally, is how to quantify subjective characteristics such as personnel accreditation. Nonetheless, it is plausible to develop different software quality certification schemes that appropriately weigh different techniques within the three approaches with respect to the criticality of the software.

### Acknowledgments

I thank Don Bagert for his efforts to keep me up to date on Texas' certification plans. ◆

### About the Author

Jeffrey Voas is a co-founder of and chief scientist for Reliable Software Technologies and is currently the principal investigator on research initiatives for the Defense Advanced Research Projects Agency and the National Institute of Standards and Technology. He has published over 85 refereed journal and conference papers. He co-wrote *Software Assessment: Reliability, Safety, Testability* (John Wiley & Sons, 1995) and *Software Fault-Injection: Inoculating Programs Against Errors* (John Wiley & Sons, 1997). His current research interests include information security metrics, software dependability metrics, software liability and certification, software safety and testing, and information warfare tactics. He is a member of the Institute of Electrical and Electronics Engineers, and he holds a doctorate in computer science from the College of William & Mary.

Reliable Software Technologies
21515 Ridgetop Circle, Suite 250
Sterling, VA 20166
Voice: 703-404-9293
Fax: 703-404-9295
E-mail: jmvoas@rstcorp.com

### References

1. Ayala, J., "Training the Microsoft Way," *Windows NT Magazine*, March 1998, pp. 122-129.
2. http://www.main.org/peboard/softweng.htm.
3. Kaner, C., "Article 2B is Fundamentally Unfair to Mass-Market Software Customers," submitted to the American Law Institute for its Article 2B review, October 1997.
4. Huang, J. C., "An Approach to Testing," *ACM Computing Surveys*, September 1975, pp. 113-128.
5. The American Law Institute and National Conference of Commissioners on Uniform Laws, Uniform Commercial Code, Article 2B (Draft), November 1997.
6. Voas, Jeffrey, "Can Clean Pipes Produce Dirty Water?" *IEEE Software*, July 1997, pp. 93-95.

# Smart Buying with the Federal Aviation Administration's Integrated Capability Maturity Model

Linda Ibrahim
*Federal Aviation Administration*

*The Federal Aviation Administration (FAA) has developed an integrated Capability Maturity Model℠ for the acquisition of software-intensive systems. This model, known as the FAA-iCMM ® [1], integrates the Systems Engineering Capability Maturity Model (SE-CMM, Version 1.1), the Software Acquisition CMM (SA-CMM, Version 1.01), and the CMM for Software (SW-CMM, Version 1.1). The FAA is achieving more effective and efficient processes and process improvement by using the integrated model rather than the three source CMMs separately. This article describes the FAA's process improvement environment, why the FAA-iCMM was constructed, the model's architecture, domain, capability levels, maturity levels, and the FAA-iCMM Appraisal Method.*

The FAA developed the FAA-iCMM to guide improvement of the engineering, management, and acquisition processes it uses to acquire software-intensive systems. Three CMMs were being used separately in different FAA directorates that work on different aspects of acquisition: the SW-CMM [2], the SE-CMM [3], and the SA-CMM [4]. These CMMs have different architectures, goals, terminology, and appraisal methods, and none alone covers all FAA system acquisition activities. Although some improvements were being made using one model, the goal of FAA-wide, full lifecycle process improvement remained elusive. In addition, the FAA had moved to using integrated product teams as the implementation arm for its new Acquisition Management System [5], and these teams needed processes that interrelated their disciplines.

The FAA-iCMM initiative began in fall 1996 with an analysis and preliminary merger of these three CMMs at the process area level. One sample process area was also elaborated at the base practice level [6, 7]. These efforts demonstrated that it was possible to integrate CMMs of different architectures and that the resultant model contained a significant reduction in the number of process areas and practices while still covering the individual CMM disciplines.

*Capability Maturity Model is a service mark of Carnegie Mellon University. CMM is registered in the U.S. Patent and Trademark Office.*

In March 1997, the FAA formed a team of FAA and external CMM and domain experts and began work on the integrated model. The project purpose was to derive a reference model that would

- Describe key elements of an effective system acquisition process.
- Describe an evolutionary improvement path.
- Have an associated appraisal method.
- Faithfully and robustly capture all features of its three source CMMs (SA-CMM, SE-CMM, and SW-CMM).

Meanwhile, the Software Engineering Institute (SEI) began to develop a Common CMM Framework (CCF) [8] to provide guidance to multiple CMM users and to assist CMM developers and integrators. The FAA-iCMM project followed those draft guidelines as they continued to evolve in parallel with FAA efforts.

A complete draft of the FAA-iCMM was completed by June 1997 and submitted to the SEI for review. FAA management adopted an FAA-iCMM-related performance goal that same month. In late September, a joint SEI-FAA review and working session was held to ensure consensus that the FAA's work captured its source CMMs and followed CMM principles, construction guidelines, and requirements as identified in the latest draft CCF documents. Version 1.0 of the FAA-iCMM was released in November 1997 with endorsement by the SEI as a new product type—an integrated Capability Maturity Model (iCMM).

## General CMM Integration Decisions

### What to Integrate (Scope)

The FAA chose to integrate the three CMMs that were already in FAA use and which together covered the engineering, acquisition, and management processes used by the FAA to acquire software-intensive systems. The Integrated Product Development CMM was briefly considered, but the draft model did not seem stable enough to be included at that time. The various drafts of SW-CMM, Version 2.0 were also coming out, but the FAA decided to use validated versions of the source CMMs to the extent possible for the initial version of the model.

### How to Represent the Model (CMM Architecture)

The FAA chose to use a hybrid architecture that includes both the continuous and staged features of its source CMMs (see Table 1). Through this "continuous with staging" architecture, the FAA-iCMM provides guidance to improve process capability and organizational maturity. As in a continuous representation, the FAA-iCMM describes the domain aspect, e.g., process areas and base practices, separately from the capability aspect (capability levels and generic practices). This feature of the continuous representation provides guidance

| FAA-iCMM (Ver. 1.0) (Continuous with Staging) | SA-CMM (Ver. 1.01) and SW-CMM (Ver. 1.1) (Staged) | SE-CMM (Ver. 1.1) (Continuous) |
|---|---|---|
| Domain Aspect: Implementation (What You Do) | | |
| Process Areas (PAs) | Key Process Areas | Process Areas |
| Purpose | Purpose | Purpose |
| Process Area Goals | Goals | |
| Base Practices (BPs) | Key Practices of the "Activities Performed" Common Feature | Base Practices |
| Process Capability: Institutionalization (How Well You Perform a Process Area) | | |
| Capability Levels | | Capability Levels |
| Capability Level Goals | | |
| Generic Practices (GPs) | Key Practices of the "Commitment to Perform" "Ability to Perform" "Measurement and Analysis" "Verifying Implementation" Common Features | Generic Practices |
| Staging: Organizational Maturity (What to Focus on Next) | | |
| Maturity Levels | Maturity Levels | |

Appraisal note: *The FAA-iCMM Appraisal Method uses process area goals and capability level goals as the major rating components during an appraisal. Maturity levels are optionally derived from capability level ratings, according to the FAA-iCMM definition of maturity level.*

Table 1. *FAA-iCMM architecture summary: architectural constructs across the source models.*

to improve any of its process areas to any capability level desired (from 1 to 5).

In addition, goals were added to process areas and capability levels. The FAA-iCMM also provides staging that groups the process areas and generic practices into maturity levels. This feature provides guidance regarding improving organizational maturity and regarding "what to focus on next" if needed. It also allows a summary rating of an organization's process maturity (from 1 to 5) if needed. For more information on architecture conversion issues, refer to [9, 10].

### Traceability

To satisfy its robustness, fidelity, and traceability requirements, the FAA-iCMM contains extensive tracing tables. These tables are at the process area level and the practice level and are included as part of each process area and base practice description. Additionally, complete mapping tables are provided in an appendix that helps readers locate where any practice in any of the source models is mapped in the FAA-iCMM (see [1]).

## Overview of the Model

The FAA-iCMM is structured to answer three process improvement questions: What activities should be performed (the domain aspect), how can performance be improved (the capability aspect), and what processes should be focused on next (maturity levels)? The FAA-iCMM Appraisal Method (FAM) supports application of the model. Each aspect is briefly described below.

### The Domain Aspect

The domain is the acquisition of software-intensive systems. There are 23 process areas derived from integrating the 52 process areas or key process areas of the three source CMMs. These process areas are grouped into four categories:

- Lifecycle or engineering.
- Management or project.
- Supporting.
- Organizational process areas.

Table 2 shows the 23 process areas of the FAA-iCMM and the major sources used to derive each process area.

Each process area description includes a purpose, goals, and from two to 10 fully elaborated base practices. Some excerpts from the Requirements Process Area (PA 02) are provided in Table 3.

### The Capability Aspect

There are five capability levels in the FAA-iCMM, and generic practices at each level provide guidance to improve any process. Generic practices are additive as process capability increases through the five levels. The capability levels, their goals, and their generic practices are summarized in Table 4.

### Maturity Levels

Maturity levels in the FAA-iCMM are groupings of process areas and generic practices. They "stage" the process areas to provide guidance to improve organizational maturity. Maturity levels are conceptually the same as capability levels, i.e., the same five levels are employed, but they provide guidance on what processes together contribute to each step of organizational maturity. Maturity levels are described in Table 5.

### Appraisal Method

FAA developed the FAM, which includes several variations. The full internal appraisal is similar to the CMM-Based Appraisal for Internal Process Improvement [11] method, except it has been adapted to a continuous model with both process area goals and capability level goals. Other appraisal types include facilitated discussion, training-based, document-intensive, questionnaire-based, interview-intensive, and external appraisal (for use by external agencies that may want to appraise the FAA's process capability).

These appraisal types draw on and adapt from several appraisal methods such as the SE-CMM Appraisal Method [12], Software Capability Evaluation [13], and Interim Profile [14]. Again, FAA's concept is to integrate and draw together

Table 2. *The integrated process areas of the FAA-iCMM.*

| FAA-iCMM Version 1.0 Process Area | Systems Engineering SE-CMM Version 1.1 Process Area | Software Acquisition SA-CMM Version 1.01 Key Process Area | Software Engineering Software-CMM Version 1.1 Key Process Area |
|---|---|---|---|
| Lifecycle or Engineering Processes | | | |
| PA01 Needs | Understand Customer Needs and Expectations | | |
| PA02 Requirements | Derive and Allocate Requirements | Requirements Development and Management | Requirements Management<br>(*Software Product Engineering) |
| PA03 Architecture | Evolve System Architecture | | (*Software Product Engineering) |
| PA04 Alternatives | Analyze Candidate Solutions | | |
| PA05 Outsourcing | Coordinate with Suppliers | Solicitation | Software Subcontract Management |
| PA06 Software Development and Maintenance | | | Software Product Engineering |
| PA07 Integration | Integrate System | | |
| PA08 System Test and Evaluation | Verify and Validate System | Evaluation | |
| PA09 Transition | | Transition to Support | |
| PA10 Product Evolution | Manage Product Line Evolution | | |
| Management or Project Processes | | | |
| PA11 Project Management | Plan Technical Effort<br>Monitor and Control Technical Effort | Software Acquisition Planning<br>Project Management<br>Project Performance Management | Software Project Planning<br>Software Project Tracking and Oversight<br>Integrated Software Management |
| PA12 Contract Management | (*Coordinate with Suppliers) | Contract Tracking and Oversight<br>Contract Performance Management | Software Subcontract Management |
| PA13 Risk Management | Manage Risk | Acquisition Risk Management | (*Integrated Software Management) |
| PA14 Coordination | Integrate Disciplines | | Intergroup Coordination |
| Supporting Processes (Not Lifecycle Phase Dependent) | | | |
| PA15 Quality Assurance and Management | Ensure Quality | | Software Quality Assurance |
| PA16 Configuration Management | Manage Configuration | | Software Configuration Management |
| PA17 Peer Review | Level 3 Common Features | | Peer Reviews |
| PA18 Measurement | Level 4 Common Features | Quantitative Process Management<br>Quantitative Acquisition Management | Quantitative Process Management<br>Software Quality Management |
| PA19 Prevention | Level 5 Common Features | | Defect Prevention |
| Organizational Processes | | | |
| PA20 Organizational Process Definition | Define Organization's Systems Engineering Process | Process Definition and Maintenance | Organizational Process Focus<br>Organizational Process Definition |
| PA21 Organizational Process Improvement | Improve Organization's Systems Engineering Process | Continuous Process Improvement | Process Change Management |
| PA22 Training | Provide Ongoing Skills and Knowledge | Training Program | Training Program |
| PA23 Innovation | Manage Systems Engineering Support Environment | Acquisition Innovation Management | Technology Change Management |

*Some of the practices in this process area contributed to the practices integrated into the FAA-iCMM process area.

**Purpose:** The Requirements process area develops requirements to meet the customer's operational need, to analyze the system and other requirements, to derive a more detailed and precise set of requirements, and to manage those requirements throughout the acquisition lifecycle.

**Goals**

1. Requirements are derived from customer needs and other appropriate sources *(BP 02.01, BP 02.02, BP 02.03, BP 02.04)*.
2. Requirements are allocated to support the synthesis of solutions *(BP 02.05)*.
3. Requirements are unambiguous, traceable, and verifiable *(BP 02.06, BP 02.09)*.
4. Requirements are controlled to establish a baseline for engineering and management use *(BP 02.07, BP 02.09)*.
5. Plans, products, and activities are kept consistent with requirements *(BP 02.08, BP 02.09)*.

***Base Practice List***

**BP 02.01 Develop detailed operational concept**: Develop a detailed operational concept of the interaction of the system, the user, and the environment that satisfies the operational need.

**BP 02.02 Identify key requirements:** Identify key requirements that have a strong influence on cost, schedule, functionality, risk, or performance.

**BP 02.03 Derive and partition requirements:** Derive and partition requirements that may be logically inferred and implied as essential to system effectiveness from the system and other, e.g., environmental, requirements.

**BP 02.04 Identify interface requirements:** Identify the requirements associated with external interfaces to the system and interfaces between functional partitions or objects.

**BP 02.05 Allocate requirements:** Allocate requirements to functional partitions, objects, people, or support elements to support synthesis of solutions.

**BP 02.06 Analyze requirements:** Analyze requirements to ensure that they can be implemented, verified, and validated by methods available to the development effort.

**BP 02.07 Capture and baseline requirements:** Capture, baseline, and place under change control the system and other requirements, derived requirements, derivation rationale, allocations, traceability, and requirements status.

**BP 02.08 Analyze and incorporate requirements changes**: Analyze all requirements change requests for impact on the product being acquired, and upon approval, incorporate the approved changes into the product, work plans, and activities.

**BP 02.09 Maintain consistency and traceability:** Maintain consistency and traceability among requirements and between requirements and plans, work products, and activities.

Table 3. *Purpose, goals, and base practice list of the Requirements process area of the FAA-iCMM.*

various appraisal methods, just as it integrated its source CMMs. All FAM variations are tailorable and cover needs for initial, interim, or full appraisal.

## Real-World Use of the Model

The FAA's CMM integration goals are to increase the efficiency and effectiveness of FAA processes and process improvement efforts. Increased efficiency is being realized by reducing the number of process areas from 52 in the separate models to 23 in the integrated model, by replacing separate training and appraisals against three CMMs with efforts against one model, and by replacing largely redundant efforts to improve similar processes with a single effort to improve an integrated process. Increased effectiveness is being realized through development of processes that cover all FAA acquisition lifecycle phases and that integrate the management, engineering, and acquisition activities of an integrated product team.

FAA management adopted the FAA-iCMM by setting an aggressive improvement goal for FAA's major software-intensive programs to achieve maturity Level 2 by December 1999 and Level 3 by December 2001. In the first year of FAA-iCMM usage, over 1,250 managers and practitioners were trained, and about 20 programs (including the targeted "major" programs, plus programs voluntarily signing up) are using the model to guide their process improvement. FAA-iCMM process improvement workshops and appraisals are finding that the model raises and promotes resolution of process integration issues across the disciplines and across the acquisition lifecycle. Working to improve the Requirements and the Transition process areas for example (both staged at maturity Level 2) has required extensive cross-directorate, cross-discipline, and cross-lifecycle participation.

A major appraisal has recently been conducted to determine interim status, to facilitate process improvement plan adjustment, and to promote even broader discussions and learning about process improvement. Meanwhile, the FAA process improvement goal is being strengthened to include new programs as they are initiated.

Other government organizations, including Warner Robins Air Logistics Center and the Internal Revenue Service, have received FAA-iCMM training and are looking toward adopting an integrated approach to process improvement. Several companies, including Lockheed Martin, have also expressed interest.

Other models may be included in future versions of the FAA-iCMM, (such as models generated from the government-industry-SEI Capability Maturity Model Integration [15] project) and other disciplines (including Human Factors and Information Security) are now being studied for inclusion. The model is available in the public domain for organizations seeking to improve their acquisition processes.

## Summary and Conclusions

CMMs provide valuable guidance to organizations committed to process improvement. When an organization needs to use multiple CMMs to cover its business needs, however, CMM-based process improvement can become costly and confusing because of the differences in CMM architecture, terminology, appraisal methods, etc. The FAA endeavored to solve this problem by integrating three CMMs into the FAA-iCMM,

| | |
|---|---|
| Level 1 – **Initial: Performed Informally** | **Description:** Base practices of the process area are generally performed.<br>**Generic Practice:**<br>1.1 Perform the process. |
| Level 2 – **Repeatable: Planned and Tracked** | **Description:** Basic management processes are established. The necessary process discipline is in place to repeat earlier successes with similar work processes. Performance of the base practices in the process area is planned and tracked.<br>**Goal:** The activities for the process are institutionalized to support a repeatable process.<br>**Generic Practices:**<br>2.1 Establish policy. 2.8 Manage configurations.<br>2.2 Allocate adequate resources. 2.9 Assess process compliance.<br>2.3 Assign responsibility. 2.10 Verify work products.<br>2.4 Ensure training. 2.11 Measure process.<br>2.5 Document the process. 2.12 Review status.<br>2.6 Plan the process. 2.13 Take corrective action.<br>2.7 Use a repeatable process. 2.14 Coordinate within the project. |
| Level 3 – **Defined: Well Defined** | **Description:** Base practices are performed according to a well-defined process using approved, tailored versions of standard documented processes.<br>**Goal:** The activities of the process are institutionalized to support a defined process.<br>**Generic Practices:**<br>3.1 Standardize the process. 3.3 Perform reviews with peers.<br>3.2 Use defined process. 3.4 Coordinate with affected groups. |
| Level 4 – **Managed: Quantitatively Controlled** | **Description:** Processes and products are quantitatively measured, understood, and controlled; detailed measures of performance are collected and analyzed.<br>**Goal:** The activities of the processes are institutionalized to support quantitative management of defined processes.<br>**Generic Practices:**<br>4.1 Establish quality objectives for product and process.<br>4.2 Select processes for measurement.<br>4.3 Select measures for the process.<br>4.4 Determine quantitative process capability.<br>4.5 Use quantitative process capability. |
| Level 5 – **Optimizing: Continuously Improving** | **Description:** Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies. A focus on widespread, continuous improvement permeates the organization. The organization establishes quantitative performance goals for process effectiveness and efficiency based on its business goals.<br>**Goal:** Continually improving processes are deployed throughout the organization.<br>**Generic Practices:**<br>5.1 Perform continual process improvement on the organizational standard and tailored processes.<br>5.2 Implement improved processes. |

Table 4. *Capability level summary.*

thereby reducing overlap and redundancies yet capturing the features of all three models. Following the latest CMM integration guidance available, the FAA-iCMM is the first proof of concept that CMM integration can work. This integrated CMM can be used to improve the processes used by system engineers, software engineers, and acquisition practitioners as they work together in integrated product teams to acquire systems. For acquisition organizations, the FAA-iCMM provides guidance for smart buying.

## Acknowledgments

The FAA-iCMM is the collaborative work of many individuals, and I acknowledge the contributions of FAA-iCMM participants including our sponsor and adviser, Art Pyster, our SEI advisers Roger Bate and Suzanne Garcia, the author team, and all our reviewers, buddies, and support staff who helped create this model. Model creation was just the beginning of our work, however, and without the support, commitment, and engagement of FAA management, process groups, and participating programs, this model would only be shelfware. Thank you for your continuing efforts to improve FAA processes, using the FAA-iCMM. ◆

## About the Author

Linda Ibrahim is the process improvement lead at the FAA. She is chairwoman of the Corporate SEPG and is the project leader, architect, and lead author of the FAA-iCMM. She is a member of the steering group for the CMM Integration effort. She worked in software engineering for more than 30 years. She was a senior member of the technical staff for several years at the SEI, and other previous employers include corporations, universities, governments, and research centers in the United States, Europe, and the Middle East. She has a bachelor's degree in mathematics from Duke University and a master's degree in information science and a doctorate in electrical engineering from the University of Hawaii.

Federal Aviation Administration
800 Independence Avenue SW
Washington, DC 20591
Voice: 202-267-7443
Fax: 202-267-5080
E-mail: linda.ibrahim@faa.dot.gov

## References

1. Ibrahim, Linda, et al., *The Federal Aviation Administration Integrated Capability Maturity Model, Version 1.0*, Federal Aviation Administration, November 1997, http://www.faa.gov/ait/sepg.
2. Paulk, Mark, et al., *Capability Maturity Model for Software, Version 1.1*, CMU/SEI-93-TR-24 and CMU/SEI-93-TR-

**Level 2 Process Areas**
**Lifecycle/Engineering Processes:** PA 01 Needs, PA 02 Requirements, PA 05 Outsourcing, PA 08 System Test and Evaluation, PA 09 Transition.
**Management/Project Processes:** PA 11 Project Management, PA 12 Contract Management.
**Supporting Processes:** PA 15 Quality Assurance and Management, PA 16 Configuration Management.
The above process areas should be at Level 2 (or higher) capability according to an FAA-iCMM appraisal.

**Level 3 Process Areas**
**Lifecycle/Engineering Processes:** PA 03 Architecture, PA 04 Alternatives, PA 06 Software Development and Maintenance, PA 07 Integration.
**Management/Project Processes:** PA 13 Risk Management, PA 14 Coordination.
**Supporting Processes:** PA 17 Peer Review.
**Organizational Processes:** PA 20 Organization Process Definition, PA 22 Training.
All Level 2 process areas plus all Level 3 PAs should be at Level 3 (or higher) capability.

**Level 4 Process Areas**
**Lifecycle/Engineering Processes:** PA 10 Product Evolution.
**Supporting Processes:** PA 18 Measurement.
All Level 2, 3, and 4 process areas of the FAA-iCMM should be at capability Level 4 (or higher).

**Level 5 Process Areas**
**Supporting Processes:** PA 19 Prevention.
**Organizational Processes:** PA 21 Organization Process Improvement, PA 23 Innovation.
All process areas of the FAA-iCMM should be at capability Level 5.

Table 5. *Maturity level summary.*


25, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., February 1993.

3. Bate, Roger, et al., *A Systems Engineering Capability Maturity Model, Version 1.1*, SECMM-95-01, CMU/SEI-95-MM-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., November 1995.
4. Ferguson, Jack, et al., *Software Acquisition Capability Maturity Model (SA-CMM), Version 1.01*, CMU/SEI-96-TR-20, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., December 1996.
5. Federal Aviation Administration Acquisition Management System, June 1997.
6. Ibrahim, Linda, "An Analysis of Three Capability Maturity Models and Their Relationship to the Acquisition Management System," Federal Aviation Administration, Technical Report, December 1996.
7. Ibrahim, Linda, "Improving Processes Across Three CMMs – Case Study Requirements Processes," Federal Aviation Administration, Technical Report, November 1996.
8. Common CMM Framework, Draft E, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., September 1997.
9. Ibrahim, Linda, "CMM Integration at the Federal Aviation Administration," *SEPG '98 Proceedings*, Chicago, Ill., March 1998.
10. Ibrahim, Linda, "The Federal Aviation Administration's Integrated Capability Maturity Model," *Systems Engineering and Software Symposium – Lockheed Martin*, New Orleans, La., May 1998.
11. Dunaway, Donna, et al., *CMM-Based Appraisal for Internal Process Improvement: Method Description*, CMU/SEI-96-TR-007, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., April 1996.
12. Garcia, Suzanne, et al., *A Description of the Systems Engineering Capability Maturity Model Appraisal Method, Version 1.0*, CMU/SEI-94-HB-05, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., June 1995.
13. Byrnes, Paul, et al., *Software Capability Evaluation, Version 3.0, Method Description*, CMU/SEI-96-TR-002, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., April 1996.
14. Hayes, Will, et al., *Interim Profile Method Description Document*, CMU/SEI-95-SR-015, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., 1995.
15. Schaeffer, Mark D., "Capability Maturity Model Process Improvement," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, May 1998.



# Need Information on Software Quality Engineering?

We have a Software Quality Engineering (SQE) Starter Kit to introduce you to the basics and an SQE Technical Report for more detailed information.

If your organization needs professional help, we can provide expert tailored consulting in any area of Software Quality. If you prefer, we offer workshops on Software Quality Assurance, Software Inspections, Moderating Inspections, Facilitated Inspection Process Definition, Defect Prevention, and Software Reliability.

You can find all this information on our Web site http://www.stsc.hill.af.mil/sqe or call us at 801-775-4399 or E-mail sqe@stsc1.hill.af.mil for any help you may need.

# Metrics for Visual Software Development

## Initial Research and Findings

Paul A. Szulewski, *Mercury Computer Systems*
Faye C. Budlong, *Draper Laboratory*

*This article provides a summary of recent research that investigated the use of visual languages (VLs) and visual programming environments (VPEs). The study reviewed the state of the practice for developing software using VLs and managing these development activities. The study concluded that there is little evidence of the use of mature practices and recommends candidate metrics for VLs and VPEs as a first step toward a method to estimate the effort required to develop software using VLs and VPEs.*

Increased demand for reliable and useful software applications has led to generations of advancements in software languages and development environments. Examples include

- The evolution from early languages, such as assembly language, to higher-order languages and fourth-generation languages.
- The development and implementation of frameworks or software engineering environments that are populated with any number of productivity tools.
- The development of graphical (or visual) front-ends for existing computer languages called VPEs.
- The development and use of VLs that allow developers to generate applications entirely within a visual environment.

Currently, the use of VPEs and VLs for general-purpose programming is undergoing such rapid adoption that it could be called a visual explosion. Applications developed using VPEs and VLs are developed rapidly and differently from applications based entirely on textual languages. It is important to understand these differences and to approach managing projects that use VPEs and VLs in a way that will allow effective project control, i.e., delivering quality software on time and within budget without interfering with the advantages inherent in the use of visual tools.

## Definitions

The following definitions are used to help form a context for the tools used to build applications visually.

- Visual Language – A computer language that uses a visual syntax, such as pictures or forms, to express programs. Text can be part of a visual syntax.
- VL Taxonomy – A system to classify VLs.
- Visual Programming – Software development that uses a visual representation of the software and allows developers to create software through managing and manipulating objects on a visual palette. Also called graphical programming.
- Visual Programming Environment – The graphical user interface (GUI) and graphical tools that are used to manage and manipulate objects on a visual palette, construct programs, interface with other software, manage the software, and execute the software.

This article provides a summary of recent research concerning the use of VLs and VPEs. The study reviewed the state of the practice for developing software using VLs and managing the development activities. Our research revealed that there is little evidence of the use of mature practices and recommends candidate metrics for VLs and VPEs as a first step toward a method to estimate the effort required to develop software using VLs and VPEs.

## Purpose of the Research

VLs and VPEs are being studied to learn how to estimate and manage software development using these new languages and environments. The goals of this initial research are to

- Identify "countables" or metrics related to VL and VPE development processes and software products.
- Develop an estimation model for software developed visually, i.e., using VLs and VPEs.

VLs and VPEs are of interest because they are presently being used to develop real applications in a range of sizes and degrees of criticality. Examples include

- GUI and GUI-related application development.
- Database search engines, e.g., visual query languages.
- Data capture and maintenance.
- Real-time data presentation.
- Space-qualified guidance, navigation, and control.
- Other real-time control systems, including aerospace and automotive applications.

With all this activity, little evidence has been found that mature practices are being used to manage development using VLs and VPEs. These types of languages are reported to be "fun to use," and the literature has yet to address the management issues that may be involved in moving from a textual model to a visual model of software development.

In addition, no evidence was found that groups using VLs and VPEs use a repeatable method to estimate development cost, effort, size, or schedule. The issues of developing large-scale applications where formal estimates and management tracking are important have only recently been addressed at any level, and the research is still in its infancy.

## Initial Results

The research focused on six specific areas:

- Finding definitions for visual languages.
- Identifying examples of commercially available VL and VPE products.
- Identifying published productivity gains and other benefits of using VLs and VPEs.
- Finding evidence of VL and VPE use in government software applications.
- Examining current VL-related measurement work.
- Identifying potential metrics for VL and VPE development.

## Examples

Tables 1 and 2 provide a limited set of examples of commercially available VLs and VPEs, respectively. The examples provide comparisons between VLs and VPEs, e.g., the output from a VPE generally is code for a specific textual programming language, and they indicate the variety of domains currently served by VLs and VPEs.

## Advantages and Disadvantages

Developing software visually has a number of advantages and disadvantages, which are summarized in the following paragraphs. Support for the advantages regarding quick results and potential increases in developer productivity are documented in the literature on visual programming. For example,

- An empirical study reports that it is easier to write programs visually than textually [1].
- Comparative studies report that there is a four to 10 times productivity gain over traditional programming techniques when working in a visual environment [2].
- A more recent empirical study concludes that visual representation improves human performance [3].

| Visual Programming Environment | Computer Language Output | Vendor | Domains |
|---|---|---|---|
| Virtual Programmer | C++ and Ada95 | VZCORP | General Purpose Component-Based Development |
| MatrixX | C, Ada95, Proprietary Scripting Language | ISI | Control Systems |
| VXP | Motif | Free from OSF | GUI Builder for X Applications |
| Café | Java | Symantec | Internet and Intranet Applications |

Table 2. *Examples of commercially available VPEs.*

### Advantages

Advantages gained through using VLs and VPEs include the following:

- They provide an opportunity for domain engineers, rather than software engineers, to develop software applications.
- Visual communication is intuitive—visual communication uses pictures rather than words (code).
- They provide quick initial results—you can examine the results sometimes within hours rather than months.
- Rather than using formal specifications to guide development, they provide a means to implement participatory development approaches using prototyping techniques and "conversations."
- They take advantage of powerful workstations and tools by providing the capability to work with pictures rather than words.
- They provide the potential to increase software development productivity.
- They provide the potential to lower lifecycle costs.

### Disadvantages

There are some potential problems that may be expected from using VLs and VPEs. These disadvantages are derived from discussions with managers and software developers who work with VLs and VPEs.

- VLs and VPEs require a new way of doing business throughout the software lifecycle, including development, test, acceptance, and maintenance—the rules have been significantly changed.
- Programmers (or software engineers) are not required; however, the quality of software produced by domain engineers may be suspect. (It is too easy to jump right in and program.)
- No industry standards are in place to control the visual languages and environments. More traditional languages, e.g., C and Ada, are standardized through concurrence of members of the software engineering community and maintenance by standards organizations. This control does not yet exist for VLs and VPEs.
- Little or no formal qualification is done for new applications because of the lack of specifications and known requirements.
- Often, especially for VLs, the bindings to other languages are weak or nonexistent.

Table 1. *Examples of commercially available VLs.*

| Language | Vendor | Domains |
|---|---|---|
| LabVIEW | National Instruments | Data Acquisition, Analysis, and Display |
| Prograph/Pictorius Net Builder | Pictorius | Macintosh Applications |
| Visual AppBuilder | Novell | Windows Server Applications |
| VEE | Hewlett Packard | Test Equipment |
| PowerBuilder | Austin Software Foundry | Windows Applications |

- Configuration control is not often considered, and the visual representations may be difficult to control using current commercially available configuration management tools.
- Development issues can be deceptively complex.
- The apparent ease of use for these tools invites abuse.

## New Development Models

New software development models are rapidly evolving as VL and VPE applications become more accepted. In general, these models are typified as being highly participatory with developers and users or other domain experts working closely to develop each application. The application tends to be its own "specification" where little upfront documentation is developed and "approved" in the traditional sense of approval.

Participatory development styles tend to involve developers, users, and other stakeholders in several ways, including

- A conversation model where the software developer and user work together with the computer to interactively build an application [4].
- "Memos and demos" that allow multiple iterations with documented output, high user visibility, and minimal specification.
- Evolutionary development using an integrated small "hot team" that consists of software developers, domain engineers, and other stakeholders to concurrently develop an application and gain approval of it.

These approaches show many similarities with rapid prototyping, including strong user (or customer) interaction during development. The software is developed, used, and refined as necessary, based on lessons learned, rather than waiting for traditional qualification or validation.

In general, formal milestones, e.g., requirements and design reviews, often are either missing or ill-defined. There are often no (or limited) formal reviews. Requirements and design are implicit in an acceptable application. Usually, the electronic design as implemented is the only representation of the application. There is often either limited or no formal testing. The project is done when the user and the developer agree that it is, or when the money runs out.

Critical government application development using VLs and VPEs have a somewhat different approach—attempts have been made to integrate evolutionary development with formal documentation and decision points. However, the concept of application development and testing appears to need further refinement. Some of the questions that should be addressed to provide confidence in these applications include

- What is a visual software "unit"?
- How detailed are the requirements?
- How do you verify software for critical applications?
- Is there a new concept of complexity?

Table 3. *Evidence of government applications using visual languages.*

| Government Agency | VL or VPE | Domain | Description |
|---|---|---|---|
| NASA/JPL | LabVIEW | Data Analysis | Telemetry Data Analyzer for Galileo Mission |
| U.S. Army | LabVIEW | Data Analysis | Graphical User Interface |
| U.S. Air Force | LabVIEW | Instrument Data Analysis | Stand-Alone Instrumentation Evaluation Tool |
| NASA/JPL | VEE | Instrument Control | Software to Support the Test of Flight Electronics |
| NASA | MatrixX | Control Systems | International Space Station |
| U.S. Air Force | Virtual Programmer Ada95 | Ada Components | Ada Joint Program Office-Sponsored VPE validation |

## Evidence of Government Use

Table 3 provides examples of government agencies that have used VLs or VPEs to help meet their software needs, the name of the VL or VPE used, the application domain in which the VL or VPE is used, and a brief description of the program or application area. Many other examples could be cited, but these provide an indication of the breadth of government applications being developed using VLs and VPEs.

## VL-Related Software Measurement

Some inroads have been made into defining measures that are applicable to VLs and VPEs. Empirical information has been gathered, as previously discussed, and some related studies have been completed. In addition, some commercial information has been developed that may be applicable to software developed visually. Examples include

- Studies such as Jeffrey V. Nickerson's "Visual Programming" [5] and E. Glinert's "Towards Software Metrics for Visual Programming" [6].
- Commercial information such as "Project Management for OO Development" [7] and "Counting a GUI Application" [8].

This information leads to the conclusion that a number of "countables" can be defined to support definition of VL metrics. Candidate countables are discussed in the next section.

## Candidate Metrics for VLs and VPEs

The countable items currently being considered as candidates for further research fall into four categories. Examples of each of these categories along with possible advantages and disadvantages follow.

### Physical Measures

Physical measures are measures of the outputs from the development effort. Those identified include the following:

- Run-time memory size, e.g., kilobytes or megabytes of memory. *Advantages*: Provides hard data that can be compared to applications

built using traditional textual languages.
*Disadvantages*: May not correlate well with effort for applications that need to be extremely efficient, e.g., real-time embedded systems with processor limitations. Size of application could grow substantially with unnecessary features, use of interpretive (rather than compiled) languages, etc.

- Processor(s) utilization, e.g., cycle time, number of cycles used, and percent of processor resources required to run an application.
  *Advantages*: Provides hard data that can be compared to applications built using traditional textual languages.
  *Disadvantages*: May not correlate well with effort for applications that need to be extremely efficient, e.g., real-time embedded systems with processor limitations. Size of application could grow substantially when processor utilization is not considered to be application critical, e.g., for data systems or other systems where memory and processing time do not need to be optimized. Interpretive language applications generally use significantly more processing resources than compiled applications, may be much easier to develop and verify, and may provide substantially less functionality when compared with compiled counterparts.
- Source lines of code (SLOC) equivalents, e.g., high-level language SLOC outputs from a VPE and functional cell contents from a spreadsheet.
  *Advantages*: SLOC are still the most used indicators of application size and allow data to be normalized based on an understandable concept. The concept of SLOC is generally understandable to software managers.
  *Disadvantages*: The concept of SLOC may not be in any way applicable to some VLs. A clear definition of SLOC needs to be used consistently to obtain consistent results. Where SLOC are automatically generated from a VPE, derived measures such as descriptiveness may not be useful or applicable. There are likely to be differences in SLOC output depending on whether the count is of automatically generated code from a VPE or hand-generated script code that may be an adjunct to the visual aspects of a VL or VPE.

### Countables in the Visual Medium

These items are entities in the physical design representation. They include

- Objects (number, semantic complexity), e.g., items on a diagram, number of diagrams, and complexity of the content of a diagram or item on a diagram.
  *Advantages*: Objects can be visually examined and counted. Within a single language or environment, object counts should yield repeatable results across several applications. This metric should help to quantify effort and schedule when combined with other measures such as number of connectors, number of interconnections, and some concept of inheritance. Some work already has been completed on complexity of applications developed with VLs.
  *Disadvantages*: May not be comparable across languages or environments. May not be easy to estimate until a design is well under way.
- Connectors (number, data complexity, control complexity), e.g., connectors between items on a diagram or indicating interfaces to items on connecting diagrams.
  *Advantages*: Connectors can be visually examined and counted. Within a single language or environment, connector counts should yield repeatable results across several applications. This metric should help quantify effort and schedule when combined with other measures such as number of objects and some concept of bandwidth.
  *Disadvantages*: May not be comparable across languages or environments. May not be easy to estimate until a design is well under way.

### OO-Related Measures

These items include measures that have been developed for object-oriented (OO) applications. There is an inherent assumption in these measures that applications developed visually use an extended concept of object orientation. Thus, the candidate measures include

- Inheritance, e.g., depth of inheritance and number of children within a class.
  *Advantages*: Can be counted in a design medium. If OO development techniques are used, will provide one of the primary OO measures of complexity.
  *Disadvantages*: Provides a secondary input to estimation needs. Provides a measure of complexity more than a measure of size. May be useful to support estimates of test effort for an OO application. VL development may not use OO techniques.
- Encapsulation, e.g., measures of how well a class (with its subclasses) provides information hiding and consistent object representation from a single (or minimal number of) source(s). Examples are lack of cohesion in methods or coupling between classes.
  *Advantages*: Measures of encapsulation provide an indication of the quality and maintainability of an OO application. Can be counted in a design medium. Also provide an indication of the effort required to test an application thoroughly.
  *Disadvantages*: Provides a secondary input to estimation needs. Provides measures of design quality, understandability, and complexity more than measures of size. May be useful to support estimates of test effort for an OO application. VL development may not use OO techniques.
- Number of interconnections, e.g., counts of “uses” and “used by” for a class or all classes within an application. Also could be counts of interfaces with external items.
  *Advantages*: Combined with number of classes in an application, provides a primary indication of application size and a “quick” estimate of application complexity. Can be counted

in a design medium. Probably most useful for estimate refinement during design. Could be useful for visual applications that do not use OO techniques.
*Disadvantages*: May not be available early enough in the software lifecycle to support effort estimation prior to the completion of a design. May be best used for estimate refinement during development or to estimate the effort required for maintenance activities.

### Function Point-Related Measures

Function points have been developed and used successfully for a number of years. Classical function points and extensions to function points could be applicable to estimates of effort for applications developed using VLs and VPEs. The applicable measures could include

- Function points, as defined in the International Function Point Users Group counting practices manual [8].
  *Advantages*: Provides a well-documented and understood approach to derive estimates of size, effort, and schedule for software applications. Can be counted in a design medium. Although function points have been shown to be useful in the information systems domain, some advocates claim that extensions, such as object points and feature points, can be adapted for OO and real-time applications.
  *Disadvantages*: May not be available early enough in the software lifecycle to support effort estimation until a reasonable amount of time has been expended on design. For maintenance, there has been little success with the development of any automated code analysis tool that can count function points in a completed application. Function point counting is complex and probably will need some adaptation for VL applications.
- Object points, e.g., counts of objects in an OO development environment.
  *Advantages*: Can be counted in a design medium. Can be used to develop size estimates for OO applications. May be useful for VL applications that do not use OO development techniques. May be useful in combination with counting objects on a visual palette.
  *Disadvantages*: May not be available early enough in the software lifecycle to support effort estimation until a reasonable amount of time has been expended on design. Counts of abstract objects and their utility to estimate VL or VPE applications is unclear.
- Feature points, e.g., extensions to function points to account for the effort required to implement algorithms for real-time applications.
  *Advantages*: Provides a well-documented approach to derive estimates of size, effort, and schedule for software applications that have real-time constraints.
  *Disadvantages*: Requires interpolation and may need to be combined with other metrics, e.g., SLOC estimates, to incorporate the algorithmic information necessary to develop cost and effort estimates. Not easy to define or implement. May not be available early enough in the software lifecycle to support effort estimation until a reasonable amount of time has been expended on design. May not be "countable" in completed applications.

## Next Steps

This research has identified a gap in the state of software development practice for estimation and measurement. Several of the practitioners of VLs and VPEs we contacted in the course of this research (including government organizations, academia, consultants, and industry) share our interest in continuing this work and have expressed the desire to form a special interest group or consortium.

We are actively seeking sponsorship and collaborators to continue this work. We have a plan to develop, using the combined expertise of our collaborators, and verify a metrics-based effort estimation model for VLs and VPEs. Once the estimation model is developed and validated, the technology will be made available to the software community at large. ◆

### Acknowledgments

This work was funded by the U.S. Air Force Embedded Computer Resources Support Improvement Program (ESIP).

During the course of this research, we knocked on many doors to obtain the information we required to perform this research. To our surprise, we found many doors open and with "welcome" signs up. We acknowledge the interest and support of

- Lt. Col. Joe Jarzombek (U.S. Air Force), ESIP director.
- Bruce Allgood, ESIP office.
- Maj. Joe Stanko (U.S. Air Force), Office of the Secretary of the Air Force for Acquisition.
- Joe Kochocki, Draper Laboratory.
- Margret Burnett, Oregon State University.
- Stan Colby, VZ Corp.
- Ed Baroth, Jet Propulsion Laboratory.
- Larry Putnam Jr., Quantitative Software Measurement.

### About the Authors

Paul A. Szulewski has held a variety of engineering-related technical and management positions since 1973. He is currently technical program manager for engineering at Mercury Computer Systems, Inc. of Chelmsford, Mass. Prior to his current position, he was at The Charles Stark Draper Laboratory, Inc. for nearly 20 years and with Sanders Associates for five years. He specializes in project planning, measurement, assessment, and process definition. He is a founding member of the National Software Council, now known as the Center for National Software Studies. He is a distinguished reviewer for *IEEE Software* and the Pentagon acquisition staff. He has pioneered research in software metrics and evaluation methods for products, processes, and organizations.

*see METRICS, page 30*

# A Model to Assess Testing Process Maturity

Ilene Burnstein, Ariya Homyen, Robert Grom, C.R. Carlson
*Illinois Institute of Technology*

*This article describes a test process assessment model based on the Testing Maturity Model[SM] (TMM) we have reported on in this publication. We discuss the test process assessment procedure, assessment inputs and outputs, the assessment questionnaire, and team selection and training criteria associated with the TMM Assessment Model (TMM-AM). Forms and tools to support test process assessment are also described, and we report on preliminary experiments with the TMM questionnaire.*

Software systems continue to have an increasingly strong impact on vital operations such as military, medical, and telecommunication systems. For this reason, it is imperative that we address quality issues that relate to both the software development process and to the software product. Our research focuses on process. We are developing a TMM designed to help software development organizations evaluate and improve their testing processes [1, 2]. Testing is applied in its broadest sense to encompass all software quality-related activities. We believe that improving the testing process thorough application of the TMM maturity criteria will have a highly positive impact on software quality, software engineering productivity, and cycle time reduction efforts.

In previous *CROSSTALK* articles (August 1996, p. 21; September 1996, p. 19), we have reported on our approach to building Version 1.0 of the TMM [1, 2]. We have also described the internal structure of the TMM, including its maturity levels, associated maturity goals, subgoals, activities, tasks, and responsibilities. In this article, we describe the TMM-AM, which is designed as a tool with which organizations may assess, evaluate, and improve their software testing processes.

## An Overview of the TMM

Development of the initial version of the TMM, as we have described in previous articles, was guided by the work done on the Software Capability Maturity Model, a process improvement model that has received widespread support from the U.S. software industry [3]. TMM, Version 1.0 has two major components [1, 2], which are discussed below.

### Set of Levels

The characteristics of each level are described in terms of organizational goals and testing capability. Each level, with the exception of Level 1, has a structure that consists of

- A set of maturity goals – these identify testing improvement goals that must be addressed and satisfied to achieve maturity at that level (Figure 1).
- Supporting maturity subgoals – these define the scope, boundaries, and needed accomplishments for a particular level.
- Activities, tasks, and responsibilities (ATR) – these address implementation and organizational adaptation issues at a specific level. Activities and tasks are defined in terms of actions that must be performed at a given level to improve testing capability; they are linked to organizational commitments. Responsibilities are assigned for these activities and tasks to three groups that represent the key participants in the testing process: managers, developers and testers, and users and clients.

*Testing Maturity Model and TMM are service marks of the Illinois Institute of Technology.*

### The Assessment Model

The TMM-AM can help organizations assess and improve their testing processes. The TMM (levels, maturity goals, subgoals, and ATRs) serves as its reference model. The outputs of a TMM assessment allow an organization to

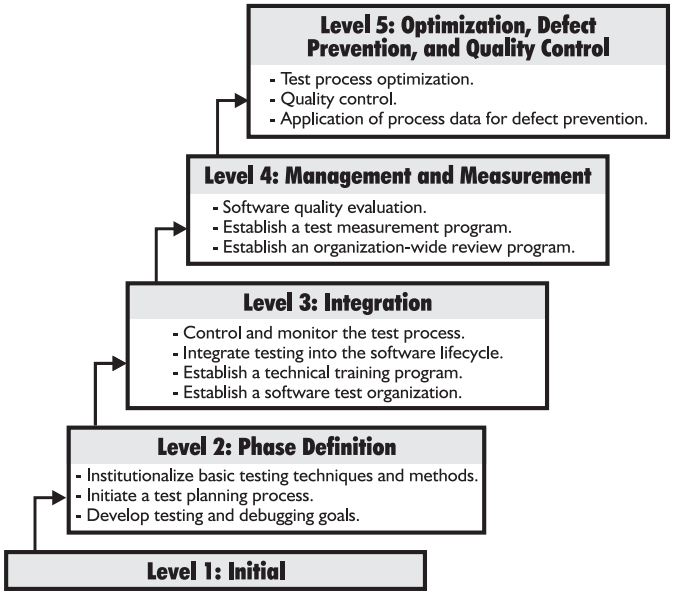
- Determine its level of testing maturity (on a scale from 1 to 5).
- Identify its testing process strengths and weaknesses.
- Develop action plans for test process improvement.
- Identify mature testing subprocesses that are candidates for reuse.

The remainder of this article discusses the TMM-AM in greater detail.

## The TMM-AM: Development Approach

The TMM-AM has the following research objectives.

Figure 1. *The 5-level structure of the TMM.*



- Provide a framework, based on a set of principles in which software engineering practitioners could assess and evaluate their software testing processes.
- Provide a foundation for test process improvement through data analysis and action planning.
- Contribute to the growing body of knowledge in software process engineering.

We have used the Capability Maturity Model (CMM) and Software Process Improvement and Capability Determination (SPICE) assessment models to guide development of the TMM-AM [3-6]. We wanted the resulting TMM-AM to be CMM Appraisal Framework (CAF) compliant [5] and integratable with the CMM assessment model so that organizations could one day perform parallel assessments in multiple process areas. A set of 16 principles has been developed to support TMM-AM design. For example, a testing process assessment model should

- Be based on a testing maturity model as its reference model.
- Support test process improvement so that an organization can achieve software product and process quality goals.
- Provide a profile of an organization's testing process capability.
- Help an organization make decisions about where to improve its testing process in order to achieve testing process maturity.
- Be integratable with other assessment models.
- Provide high-quality data and repeatable, reliable results.
- Provide visibility to the testing process.

## The TMM-AM Components

Based on the 16 principles, the CMM assessment model, SPICE, and the CAF [3-6], we have identified a set of inputs and outputs and have developed a set of three components for the TMM-AM:

- The assessment instrument (a questionnaire).
- The assessment procedure.
- Team training and selection criteria.

A set of inputs and outputs is also prescribed for the TMM-AM. The relationship between these items is shown in Figure 2. A discussion of the components follows.

### The Assessment Procedure

The TMM-AM assessment procedure consists of a series of steps that guide an assessment team in carrying out a testing process self-assessment. The principle goals are to

- Ensure the assessment is executed with efficient utilization of the organization's resources.
- Guide the assessment team as to who to interview and how to collect, organize, and analyze assessment data.
- Support the development of a test process profile and the determination of a TMM level.
- Guide the assessors in developing action plans for test process improvement.

A brief summary of the steps in the assessment procedure follows:

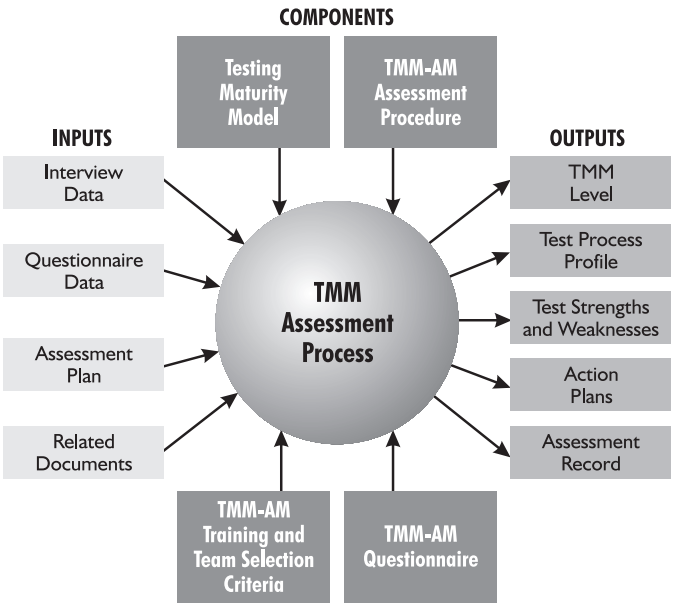



Figure 2. *The TMM assessment process: components, inputs, and outputs.*

### Preparation

This includes selecting and training the assessment team, choosing the team leader(s), developing the assessment plan, selecting the projects, and preparing the organizational units that are participating in the assessment. A statement of assessment purpose, scope, and constraints is also prepared to guide the development of the assessment plan.

### Conducting the Assessment

The team collects and records assessment information from interviews, presentations, questionnaires, and relevant documents. All collected information must be protected by a confidentiality agreement. The TMM level of the organization is determined by analysis of the collected data and use of a ranking algorithm.

Our TMM-AM ranking algorithm is similar to the algorithm described by S. Masters, et al., in their work on the CAF [5]. First, it requires a rating of the maturity subgoals, then the maturity goals, and finally the maturity level [7]. Our "degree of satisfaction" measure with respect to the maturity subgoals and goals is more fine-grained than the corresponding measure in the Masters model. Our purpose was to provide more detailed information to identify test process strengths and weaknesses. We also provide guidance for prioritization of goal areas needed for test process improvement.

### Reporting the Assessment Outputs

The TMM-AM outputs include a process profile, a TMM level, a statement of test process strengths and weaknesses, and the assessment record. The assessment team prepares the process profile, which gives an overall summary of the state of the organization's testing process. The profile is based on analysis of the assessment data and results of the ranking process. The profile can be presented as a graphical display or in the form of a matrix that indicates maturity goals and subgoals that are

satisfied, not satisfied, not applicable, or not rated. The profile also includes the TMM level, a summary of test process strengths and weaknesses, and recommendations for improvements.

The assessment record is also completed in this step. This written account includes

- Names of assessment team members.
- Assessment inputs and outputs.
- Actual schedules and costs.
- Tasks performed.
- Task durations.
- People responsible.
- Data collected.
- Problems that occurred.

The assessment outputs can be delivered as a presentation or a written report (the final assessment report) or both.

### Analyzing the Assessment Outputs

The assessment team uses the assessment outputs to identify and prioritize goals for improvement. An approach to prioritization is described in [7]. Quantitative test process improvement targets need to be established in this phase so they can support the action plans developed in the next step.

### Action Planning

An action planning team develops action plans that focus on improvements in the high-priority areas identified in the previous step. The action planning team can include assessors, Software Engineering Process Group members, software quality assurance staff, or opinion leaders chosen from among assessment participants [8]. Inputs to action planning include the final assessment report, the process profile, and prioritized areas for improvement.

The action plan describes specific actions needed to improve existing practices (and to support the addition of missing practices) so the organization can move to the next TMM level. The action plan, like all other software engineering project plans, should include measurable goals, tasks, responsibilities, resources required, risks and benefits, and reporting and tracking mechanisms. Action planning can be accomplished through the convening of a workshop directed by the action planning team. The result should be a draft of an action plan. The workshop members should also identify pilot projects that will implement the new process.

### Implementing Improvement

Developed and approved action plans can be applied to selected pilot projects, which are monitored and tracked to ensure task progress and achievement of target goals. Favorable results set the stage for organizational adaptation of the new process.

### The TMM Assessment Questionnaire

Assessment instruments are needed to help collect and record assessment information, maintain a record of results, and provide information for assessment postmortem analysis. We use the questionnaire as our assessment instrument because it

- Supports CAF compliance.
- Facilitates integration with other process assessment instruments.
- Ensures assessment coverage of all activities, tasks, and responsibilities identified in each maturity goal for each level of the TMM.
- Provides a solid framework in which to collect and store assessment data.

Our choice was also influenced by the success of the CMM questionnaire as an assessment instrument [3]. The TMM questionnaire consists of eight parts:

- Instructions for use.
- Respondent background.
- Organizational background.
- Maturity goal and subgoal questions.
- Testing tool use questions.
- Testing trends questions.
- Recommendations for questionnaire improvement.
- A glossary of testing terms [4, 7].

Components 2 and 3 of the questionnaire gather information about the respondent, the organization, and the projects that will be involved in the TMM assessment. Maturity goal and subgoal questions in component 3 are organized by TMM Version 1.0 levels, and include a developer or a tester, a manger, and a client or a user view. The questions determine to what extent the organization has in place mechanisms to achieve the maturity goals and resolve maturity issues at each TMM level. The responses are input to the ranking algorithm that determines a TMM level.

The testing tool component records information about type and frequency of tool use. This information can help the action planning team make recommendations for future tool usage. We added the testing trends section to provide a perspective on how the testing process in the organization has evolved over the last several years. This information helps the assessment team prepare the assessment profile and assessment record.

The recommendations component allows each respondent to give the TMM-AM developers feedback on the clarity, completeness, and usability of the questionnaire. A complete TMM questionnaire is found in [7]. The questionnaire can also be found on the Web site noted in the "Forms and Tools for Assessment Support" section of this article.

### Assessment Training and Team Selection Criteria

Self-assessment of your organization's testing process requires a trained assessment team, the members of which are selected from within the organization [7]. Team members should be selected in a manner that ensures that they understand assessment goals, have the proper knowledge experience and skills, have strong communication skills, and are committed to improving the testing process. Assessment team size should be appropriate for the purpose and scope of the assessment.

We have adapted SPICE guidelines to select and prepare an effective assessment team [6, 7]. Preparation is conducted by the assessment team leader who is experienced in TMM assessments. Preparation includes topics such as an overview of the TMM, process management concepts, interviewing techniques, data collection, and data analysis. Training activities include team-building exercises, a walk-through of the assessment process, filling out a sample questionnaire, performing data analysis, and learning to prepare final reports.

## Forms and Tools for Assessment Support

We have developed several forms and templates and a tool that implements a distributed version of the TMM questionnaire to support a TMM assessment team [7, 9]. These tools are important to ensure the assessments are performed in a consistent, repeatable manner, to reduce assessor subjectivity, and to ensure the validity, usability, and comparability of the assessment results. Tools and forms also help to collect, formalize, process, store, and retrieve assessment information. The tools and forms we have developed include the Process Profile and Assessment Record forms, which have been described in previous sections of this article, and also include

- Team Training Data Recording Template – This allows the team leader to record and validate team training data. This data can be used in future assessments to make any needed improvements to the assessment training process.
- Traceability Matrix – This matrix is filled in as assessment data is collected, allows the assessors to identify sources of data, resolve data related issues, and ensure the integrity of the data.
- Web-Based Questionnaire – A complete version of the TMM-AM questionnaire is at http://www.csam.iit.edu\~tmm. The Web-based questionnaire was designed so that assessment data could easily be collected from distributed sites and organized and stored in a central data repository that could be parsed for later analysis [9]. Developed using an HTML-based development tool, it runs on multiple operating systems, allowing data collection from users around the world, thus providing support for test process assessment to local and global organizations. A detailed description of tool development is given [9]. The Web-based questionnaire and links to supporting information related to the TMM is found at the above Web site. We welcome comments and recommendations.

## Preliminary Results on Questionnaire Usage

Two software engineers from different development organizations have evaluated the TMM questionnaire and have applied it to three development groups in their organizations (one engineer evaluated two groups). Their feedback helped revise and reorganize some TMM questions, experiment with our ranking algorithm using actual industrial data, generate sample action plans, and study problems of testing process improvement in real-world environments.

Obtaining and analyzing this industrial data, although on a small scale, has been useful to our research team. One interesting result was that all three groups were evaluated to be at TMM Level 1, but strengths and weaknesses of each group were significantly different. Two groups satisfied several maturity goals at the higher levels of the TMM. Given the quality of the existing processes for the latter two groups, they should be able to reach TMM Level 2 in a relatively short time. More experimental data is needed to further test the usefulness and effectiveness of the TMM and the Assessment Model for test process assessment and improvement.

## Future Plans

Our future plans include research on formal integration of TMM and CMM components so that organizations can carry out parallel assessments in several process areas. We also are planning the development of more intelligent tools to aid the assessors. Wider industrial application of the TMM-AM is planned to help us evaluate its usefulness and effectiveness for test process improvement. ◆

### About the Authors

Ilene Burnstein holds a doctorate from Illinois Institute of Technology, where she is an associate professor of computer science, teaching undergraduate and graduate courses in software engineering. Her research interests include software process engineering, testing techniques and methods, automated program recognition and debugging, and software engineering education.

Computer Science Department
Illinois Institute of Technology
10 West 31st Street
Chicago, IL 60616
Voice: 312-567-5155
Fax: 312-567-5067
E-mail: csburnstein@minna.iit.edu

Ariya Homyen is a doctoral candidate in computer science at Illinois Institute of Technology. She has a master's degree in computer science from the University of New Haven and a bachelor's degree from Chulalongkorn University in Thailand. Upon completing her doctorate, she will return to her position at the Ministry of Science, Technology, and Energy in Thailand. Her research interests include test process improvement, test management, and process reuse.

Computer Science Department
Illinois Institute of Technology
10 West 31st Street
Chicago, IL 60616
Voice: 312-567-5150
Fax: 312-567-5067
E-mail: homyari@minna.iit.edu

Robert Grom has a bachelor's degree from Southern Illinois University and a master's degree in computer science from Illinois Institute of Technology. He has worked as a hardware engineer and is currently manager of data collection software for SAFCO Technologies, where he develops software for cellular test equipment. His research interests are software testing, test process improvement, and data communications.

SAFCO Technologies, Inc.
6060 N. Northwest Highway

Chicago, IL 60631
Voice: 773-467-2673
Fax: 773-594-2618
E-mail: rag@safco.com

C.R. Carlson holds a doctorate from the University of Iowa. He is a professor in the computer science department at Illinois Institute of Technology. He has published extensively in the fields of database design, information architecture, and software engineering. His research interests include object-oriented modeling, design and query languages, and software process issues.

Computer Science Department
Illinois Institute of Technology
10 West 31st Street
Chicago, IL 60616
Voice: 312-567-5152
Fax: 312-567-5067

## References

1. Burnstein, I., T. Suwanassart, and C.R. Carlson, "The Development of a Testing Maturity Model," *Proceedings of the Ninth International Quality Week Conference*, San Francisco, May 21-24, 1996.
2. Burnstein, I., T. Suwanassart, and C.R. Carlson, "Developing a Testing Maturity Model," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah; Part I: August 1996, pp. 21-24; Part II: September 1996, pp. 19-26.
3. Paulk, M., C. Weber, B. Curtis, and M. Chrissis, *The Capability Maturity Model: Guideline for Improving the Software Process*, Addison-Wesley, Reading, Mass., 1995.
4. Zubrow, D., W. Hayes, J. Siegel, and D. Goldenson, "Maturity Questionnaire," Technical Report, Software Engineering Institute, CMU/SEI-94-SR-7, June 1994.
5. Masters, S. and C. Bothwell, "A CMM Appraisal Framework, Version 1.0," Technical Report, Software Engineering Institute, CMU/SEI-95-TR-001, February 1995.
6. ISO/IECJTC1/WG10, "SPICE Products," Technical Report, Type 2, June 1995.
7. Homyen, A., "An Assessment Model to Determine Test Process Maturity," Diss., Illinois Institute of Technology, July 1998.
8. Puffer, J. and A. Litter, "Action Planning," *IEEE Software Engineering Technical Council Newsletter*, Vol. 15, No. 2, pp. 7-10.
9. Grom, R., "Report on a TMM Assessment Support Tool," Technical Report, Illinois Institute of Technology, April 1998.

---

*METRICS, from page 25*

Mercury Computer Systems
199 Riverneck Road
Chelmsford, MA 01824-2820
Voice: 978-256-0052 ext. 320
Fax: 978-256-3599
E-mail: paulski@mc.com
Internet: http://www.mc.com

Faye C. Budlong is a principal member of the technical staff at The Charles Stark Draper Laboratory, Inc. in Cambridge, Mass. She provides expertise in software standards development and application, software product evaluations, software requirements analysis, standard-compliant software, and document development within Draper Laboratory. She has a bachelor's degree in mathematics from Roger Williams University in Bristol, R.I. and a master's degree in education from Northeastern University in Boston, Mass.

The Charles Stark Draper Laboratory, Inc.
555 Technology Square
Cambridge, MA 02139
Voice: 617-258-2054
Fax: 617-258-3939
E-mail: budlong@draper.com
Internet: http://www.draper.com

## References

1. Pandey, R.K. and M. Burnett, "Is It Easier to Write Matrix Manipulation Programs Visually or Textually? An Empirical Study," *Proceedings of the 1993 IEEE Symposium on Visual Languages (VL93)*, 1993, pp. 344-351.
2. Baroth, E. and C. Hartsough, "Visual Programming in the Real World," *Visual Programming*, M. Burnett, et al., eds., Manning Publications, 1995.
3. Whitley, K.N., "Visual Programming Languages and the Empirical Evidence For and Against," *Journal of Visual Languages and Computing*, October 1996.
4. Baroth, E. and C. Hartsough, "Visual Programming Improves Communication Among the Customer, Developer and Computer," Presentation at National Instruments User Symposium, 1995.
5. Nickerson, J.V., *Visual Programming*, Diss., New York University, New York, N.Y., 1994.
6. Glinert, E., "Towards Software Metrics for Visual Programming," *International Journal of Man-Machine Studies*, Vol. 330, Academic Press, 1989, pp. 425-445.
7. *Project Management for Object-Oriented Development*, Austin Software Factory, Austin, Texas, 1996.
8. *Function Point Counting Practices Manual*, International Function Point Users Group, Waterville, Ohio, Version 4, January 1994.

## Note

1. Domain engineers are subject matter experts and may include, for example, mathematicians and control engineers.

CROSSTALK

| | |
|---|---|
| **Sponsor** | ***Lt. Col. Joe Jarzombek***<br>801-777-2435 DSN 777-2435<br>jarzombj@software.hill.af.mil |
| **Publisher** | ***Reuel S. Alder***<br>801-777-2550 DSN 777-2550<br>publisher@stsc1.hill.af.mil |
| **Managing Editor** | ***Forrest Brown***<br>801-777-9239 DSN 777-9239<br>managing_editor@stsc1.hill.af.mil |
| **Senior Editor** | ***Sandi Gaskin***<br>801-777-9722 DSN 777-9722<br>senior_editor@stsc1.hill.af.mil |
| **Graphics and Design** | ***Kent Hepworth***<br>801-775-5798<br>graphics@stsc1.hill.af.mil |
| **Associate Editor** | ***Lorin J. May***<br>801-775-5799<br>backtalk@stsc1.hill.af.mil |
| **Editorial Assistant** | ***Bonnie May***<br>801-777-8045<br>editorial_assistant@stsc1.hill.af.mil |
| **Features Coordinator** | ***Denise Sagel***<br>801-775-5555<br>features@stsc1.hill.af.mil |
| **Customer Service** | 801-775-5555<br>custserv@software.hill.af.mil |
| **Fax** | 801-777-8069 DSN 777-8069 |
| **STSC On-Line** | http://www.stsc.hill.af.mil |
| **CROSSTALK On-Line** | http://www.stsc.hill.af.mil/Crosstalk/crostalk.html |
| **ESIP On-Line** | http://www.esip.hill.af.mil |

***Subscriptions:*** Send correspondence concerning subscriptions and changes of address to the following address:

Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205

E-mail: custserv@software.hill.af.mil
Voice: 801-775-5555
Fax: 801-777-8069 DSN 777-8069

***Editorial Matters:*** Correspondence concerning *Letters to the Editor* or other editorial matters should be sent to the same address listed above to the attention of *CROSSTALK Editor* or send directly to the senior editor via the E-mail address also listed above.

***Article Submissions:*** We welcome articles of interest to the defense software community. Articles must be approved by the *CROSSTALK* editorial board prior to publication. Please follow the *Guidelines for CROSSTALK Authors*, available upon request. We do not pay for submissions. Articles published in *CROSSTALK* remain the property of the authors and may be submitted to other publications.

***Reprints and Permissions:*** Requests for reprints must be requested from the author or the copyright holder. Please coordinate your request with *CROSSTALK*.

***Trademarks and Endorsements:*** All product names referenced in this issue are trademarks of their companies. The mention of a product or business in *CROSSTALK* does not constitute an endorsement by the Software Technology Support Center (STSC), the Department of Defense, or any other government agency. The opinions expressed represent the viewpoints of the authors and are not necessarily those of the Department of Defense.

***Coming Events:*** We often list conferences, seminars, symposiums, etc., that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the *CROSSTALK* Editorial Department.

***STSC On-Line Services:*** STSC On-Line Services can be reached on the Internet. World Wide Web access is at http://www.stsc.hill.af.mil. Call 801-777-7026 or DSN 777-7026 for assistance, or E-mail to schreifr@software.hill.af.mil.

***Publications Available:*** The STSC provides various publications at no charge to the defense software community. Fill out the Request for STSC Services card in the center of this issue and mail or fax it to us. If the card is missing, call Customer Service at the numbers shown above, and we will send you a form or take your request by phone. The STSC sometimes has extra paper copies of back issues of *CROSSTALK* free of charge. If you would like a copy of the printed edition of this or another issue of *CROSSTALK*, or would like to subscribe, please contact the customer service address listed above.

The ***Software Technology Support Center*** was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies that will improve the quality of their software products, their efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery. *CROSSTALK* is assembled, printed, and distributed by the Defense Automated Printing Service, Hill AFB, UT 84056. *CROSSTALK* is distributed without charge to individuals actively involved in the defense software development process.



# The New Terrorists

If you're having problems with snakes coming to get you from behind your bedroom chair at night, it helps to turn up the lights, open the door a crack, and squeeze the stuffing out of your Tickle-Me-Ernie doll. Just ask my two-year-old son, Daren. He still doesn't know where his dreams stop and reality begins, but he feels much safer since we instituted these powerful anti-snake defenses.

Thankfully, unlike toddlers, we adults can separate fantasy from reality. For example, a few years ago, a movie about computer cracking and sabotage called "*The Net*" came out. It was packed with eye rollers, but these were quickly rebutted by Internet chat forums in one huge collective "Puh-LEEZE!"

The first clue about the movie's realism was that the lead character, a lonely geek beta tester, was played by the lovely Sandra Bullock—a casting decision equivalent to making a movie about the Miss America Pageant with the lead, Miss Delaware, played by Wilford Brimley. (Not that the cyberculture—which likely includes readers of this journal—isn't full of attractive, fascinating people who are neither sensitive to negative stereotypes nor vindictive toward those who propagate these stereotypes. Ha-ha! Please leave my medical records alone!)

However, it was mostly the technical issues that made net surfers guffaw at "*The Net.*" For example, Bullock's character routinely accesses an advanced multimedia Internet full of cutesy features unavailable to the general public at approximately 1,153 times the bandwidth of typical modems. And get this: The bad guys manage to steal vast sums and even kill people by breaking into critical banking, police, hospital, and air-traffic computers.

Ha-ha! *Hacker terrorists*? What planet do these Hollywood types live on, where critical computer systems are even indirectly connected to the Internet, opening the door for terrorist geeks to remotely break in and cause havoc?

Well, okay, the world is spending billions of dollars each year to allow exactly that. That's why I wanted to see if cyberterroism were for real or just a hyped-up Hollywood dream. What I saw made my head spin like an unbalanced Maytag.

After a few clicks in *Yahoo!* I was visiting sites with step-by-step instructions on how to slip past firewalls, steal passwords, tap into phone and data lines, and cover your tracks. Plus, there were various free "cracking" tools available for download. Purveyors of this information seemed proud of the ease with which they allegedly find weak links and holes in supposedly secure systems, where they could cause serious damage if they were criminally inclined. (Which, of course, they never are! Please don't double my bank account balance!)

Speaking of which, I also read news reports on several successful electronic bank break-ins, including a partially successful $10 million heist. And according to the head of a major U.S. media organization, a team of hired government crackers last year showed what kind of damage organized terrorists could do. Using only techniques found on the Internet, they allegedly broke into "secure" computers and made power grids fail, air traffic control systems go haywire, oil refinery pumps stop working, and they compromised supply networks. They supposedly covered their tracks well enough that the victims wouldn't acknowledge being cracked—these were considered unexplainable glitches, not attacks.

So as we blithely barge headlong into a world where every critical computer system is in some way connected to the Internet—I suppose someone is already working on a method to remotely pilot oil tankers over the Web—I wonder how often we're stopping to ask the following questions.

- Just because a system *can* have a Web interface, does that mean it *should*?
- If a critical systems is accessible to anyone with a Web browser and password, why do crackers snicker so loudly when such a system is declared "secure"?
- Could evil crackers rig it so that the Miss America Pageant was actually won by Wilford Brimley? Would this help resolve the swimsuit debate?

These tough questions impact all of us. And the fuzzy line between fact and fiction makes me wonder: How real and dangerous are terrorist "cybersnakes"? Are our defenses good or are we counting on "Ernie" to protect us? We must address these questions, or later we may have tougher questions to answer. For example: mascara or no mascara for Miss Delaware's back hair? – Lorin May

*Got an idea for BACKTALK? Send an E-mail to* backtalk@stsc1.hill.af.mil