

Software Quality Matters

Newsletter of The Software Quality Institute at The University of Texas at Austin in Cooperation with the Austin Software Council and the Austin Software Process Improvement Network

Volume 6, Number 1



Spring 1998

SQI Team Helps Austin Quality Council Develop Software Quality Award Criteria

by Gary Cobb

Yes, we've taken another swing at developing software quality award criteria for the Austin Quality Council, and this time we've done it!

This year, Austin-area companies and organizations will be able to apply for the Austin Quality Council's Malcolm Baldrige-based quality award under software-specific award criteria thanks to the hard work of many Austin-area software professionals.

Only Two Tasks Remain

Joining me in leading the software-award-criteria development effort was IBM's Marilyn Payne. Now, only two more tasks remain before companies can apply in April 1998 for the Greater Austin Quality Award (GAQA) in software development.

The first task is the final review of the new criteria. Herb Krasner, chair of the SQI Advisory Group, and Geree Streun, chair of the Austin Software Process Improvement Network (A-SPIN), will perform this review.

Second, Marilyn and I will tackle integrating the scoring mechanism for the software-specific criteria with those that are currently being used for the Malcolm Baldrige scoring in order to form the 1998 GAQA scorebooks for each level.

Spring Announcements Expected

More than 500 Austin area software development businesses or organizations within larger companies will be eligible to apply for this award. By doing so, these companies or organizations should be able to make more meaningful gains in their software quality improvement efforts through applying annually.

New applications are expected to be announced at the February 19 A-SPIN meeting and at the spring Greater Austin Quality Symposium, sponsored by the Austin Quality Council on April 8.

If your company would like an independent examination of your software processes based upon these new criteria, call Jim Nelson, director of the Greater Austin Quality Council, at 322-5603 or download the criteria from the SQI website: www.utexas.edu/coe/sqi.

The Long Row

We've hoed a long row in developing these award criteria. In 1996, a committee led by Geree Streun, which included Celeste Armstrong and me, delivered a strawman set of award criteria

to the Greater Austin Quality Council. The council, however, determined that it didn't have time to publish the criteria and train enough examiners for one-day site visit examinations.

Then, in 1997, the Malcolm Baldrige criteria themselves changed. The council found that it had its hands full implementing these changes and couldn't incorporate the software criteria too.

Finally, this year, the time was right. The Austin Quality Council is now ready to add software-specific criteria to its Malcolm Baldrige-based applications. In short, 1998 will be the first year that both the applications and the examinations will include content focused on strengthening the capabilities of software development organizations.

This Year's Strategy

The strategy taken by this year's committee is to form a "lite" version of the most effective tool known to the software industry for quality improvement: the Capability Maturity Model (CMM), version 1.1, from the Software Engineering Institute (SEI). This model, which has been in public use for nearly six years, must be tailored for this application, however, since a CMM-based assessment typically takes more than a week to perform. Under our CMM lite approach, we will complete our examination during a one-day site visit to the software development business or organization.

GAQA applications at levels 1, 2, and 3 will include software capability criteria from the CMM at levels 2, 3, and 4, respectively.

GAQA (continued on p. 2)

What's Inside

SQI Symposium	2
Software as Product Line, Part II	5
Spring SQI Seminars	6
Cost of Quality and the ROI	7
SQA and the Dodo Bird	10

Second Annual SQI Symposium on Software Quality Slated for April 2nd

Software Metrics for Software Quality— Theme of Second Annual SQI Symposium

by Cinda Cyrus



Internationally recognized speakers from industry and academia will offer their insights and predictions about the relationship between software metrics and software quality on April 2 in Austin at the second annual SQI Symposium.

The full-day meeting will be held in the Commons Auditorium at UT's Pickle Research Campus. Because of limited seating, early registration is encouraged.

The following topics and speakers are planned.

- A Capability-based Software Organization: Building Core Competencies in Software—Victor R. Basili, keynote speaker
- Why Most Software Quality Metrics Do Not Measure Software Quality—Norman Fenton
- Software Forensics: Digging through the Data—Shari Lawrence Pfleeger
- The Cost of Software Quality: A New Metric Whose Time has Come—Herb Krasner
- Using Software Failure Analysis Data to Motivate Change—Bob Grady
- Politics and Polemics in a Corporate

Software Measurement Program—
Bill Curtis

A Capability-based Software Organization

In his keynote address, Basili will identify various software capabilities as core competencies that are tailored to particular businesses and define the software capabilities needed to support strategic goals of the corporation.

These concepts involve defining and integrating business goals and quantifiable software quality goals, supporting the selection of the appropriate methods and tools for achieving those goals, evaluating and feeding back information that supports the current project, and learning for improvement on future projects.

Basili is a professor in the Department of Computer Science and director of the Institute for Advanced Computer Studies at the University of Maryland.

Why Most Software Quality Metrics Don't Work

By drawing on recent case study data from major industrial systems, Fenton will show that some of the most widely held beliefs

about software metrics are demonstrably wrong. In discussing some of the apparent paradoxes in the software metrics field, Fenton will explain how some basic additional data are needed for good metrics and predictive models.

Fenton is professor of Computing Science at the Centre for Software Reliability, City University, London, and director of the Centre for Software Reliability Ltd.

Software Forensics

While many of our software quality efforts depend on examining information related to software faults and failure, Pfleeger believes that often the data collected do not always match our needs. She will discuss what we would like to collect, what we actually do collect, and how to narrow the gap between the two.

Pfleeger is president of Systems/Software, Inc., and professor and director of Howard University's Center for Research in Evaluating Software Technology.

Cost of Software Quality

In his presentation, Krasner will discuss how the emerging measurement technique

Symposium (continued on p. 8)

GAQA (continued from p. 1)

Applying for the Award

All applicants for the award, even those in pure software development, must complete the portion of the application that addresses the Malcolm Baldrige criteria. This means that a normal Malcolm Baldrige-based evaluation must be done on the application and that the examiners of this portion of the application must receive the normal training on the criteria and follow the standard approach in scoring the application.

Software development businesses or organizations must also address the CMM lite criteria in an add-in section of their application. Rather than requiring that all the level-dependent key process areas (KPA's), with their practices and common features, be addressed in the award application, and in the one-day site visit, the CMM lite criteria require that the applicant only state how the organization performs each of the KPA's.

The site visit examination will employ a variable subset of questions based on the CMM key practices and common features applicable to the level of the application. A score on the questions will be combined with the score of the Malcolm Baldrige criteria to establish the composite score for the GAQA. Award-winning software development organizations will also receive examiner feedback in the final reports that will be delivered at the awards banquet.

Requirements for Examiners

Three to five trained examiners comprise today's examining team. We have recommended two changes to the examination process when the CMM lite criteria are instituted. First, we propose that at least one examiner for level-1 applicants, two examiners for level-2 applicants, and three examiners for level-3 applicants work in software organizations. We also recommend that the training for examiners be extended in order to cover how to tailor the CMM-based questions to be used in the site visit and how to calculate the composite score.

Becoming an Examiner

If you are a skilled software professional—especially if you have had some experience with the CMM version 1.1—and you would like to become an examiner this year, then please call Jim Nelson at the Greater Austin Quality Council at 322-5603 for an examiner's application form.

About the author. A member of SQI's Advisory Group, Gary Cobb works at Dell Computer Corporation and teaches multimedia courses on systems design and authoring at Southwest Texas State University. He is a member of the SQI Advisory Group and the A-SPIN board and is co-chair of the IEEE Multimedia Systems '98 Conference to be held June 27-July 1.

Developing Software as a Product Line is Key Development Paradigm for the New Millennium

by Paul Clements



In Part I of this article, which appeared in the previous edition of Software Quality Matters, the author described how CelsiusTech Systems AB launched a software product line and introduced the Software Engineering Institute (SEI) Product Line Systems Program and its work in helping organizations adopt a product-line approach.

Now, in Part II, the conclusion of this article, the author discusses issues other than technology that a company must master if it is to adopt product line development and how the transition is made to this new development paradigm.

Mastering Other Business Issues

Technology is not the only barrier to successful product line adoption. In pursuing its goal of identifying the best practices for product line production and helping organizations negotiate the hurdles of adopting a product-line approach, the Software Engineering Institute (SEI) Product Line Systems Program has learned that mastering organization, process, and business issues are equally vital.

Financial. A major business issue is financial. How do organizations justify the investment cost for building generic components that will populate the product line systems?

While investments in software artifacts have been considered “sunk” costs in the past, in the product line paradigm they are investment costs; architecture, component designs, and other assets will be reused across many systems, amortizing their costs. Organizations must establish cost models to help them understand, manage, and exploit investment opportunities.

Organization. Traditional organizational structures that simply have one business unit per product are not appropriate for product lines. In addition there should be a unit responsible for maintaining the core reusable assets—the architecture, the reusable components, and so forth. If these assets are under the control of a business unit associated with one product or one large customer, the assets will evolve to serve that business unit, that product, and that customer, to the exclusion of the others.

Establishing a business unit to manage reusable assets means establishing processes for making the assets satisfy the needs of all of the business units that use them. This is a crucial role that requires staff skilled in abstraction, in design, and in negotiation and creative problem solving.

Customer Management. An important product line function is customer management. Customers interact with a product line organization in a different way. Marketers can no longer agree to anything the customer wants, but must instead nudge the customer to set requirements that can be fulfilled by a version of the product line within the planned scope of variation. Contrary to intuition, this often makes the customer much happier than before.

Under the new paradigm, the marketer can point to specific requirements that would put the customer’s new system outside the scope of the product line, which would increase cost and delivery time, lower system reliability, and keep that customer out of a community of customers to which the vendor pays a lot of attention.

Thus, the customer could clearly (and probably for the first time) see the real cost of those *special* requirements and make an informed decision about their real value. Most often, a variant of the standard, or product-line, system will do just fine.

The customer community should not be underestimated. In CelsiusTech’s case, its naval customers around the world banded together to form a user’s group. They did this in their own self-interest to provide a forum in which they could jointly derive new requirements for their evolving systems and drive CelsiusTech to supply new systems more economically than it otherwise might. Of course it doesn’t take much to realize how beneficial this is to CelsiusTech as well: its customer base is jointly defining next-generation products and effectively buying in to the CelsiusTech approach, thus guaranteeing the vitality of the product line for years to come. In the process, CelsiusTech became more like a commercial vendor than a large-system job shop, and it had to manage that reshaping.

Processes. Processes must, of course, be updated to deal with product line development. For instance, configuration management becomes crucial when several different versions of several different components to supply several different systems must be buildable. And what happens when the components change? Do you go back and retrofit all of the earlier systems, or do you let them go their own way? These and other questions must be carefully considered.

The Transition to Product Line Development

Successful transition to product line development is, thus, a careful blend of technology, process, organization, and business factors improvement. The Product Line Systems Program is attempting to codify these practices and understand how they vary with the type of organization involved and the kind of systems being built. Through a series of workshops, case studies, and collaborative engagements, we are helping build a community of organizations interested in moving to a product line approach for their software products.

Looking Ahead

The history of programming can be viewed as an upward spiral in which the abstractions manifested by components have grown larger and more application meaningful, with resulting increases in the reuse and applicability of those components. From subroutines in the ‘60s, to modules in the ‘70s, to objects in the ‘80s, to component-based systems in the ‘90s, software product lines will raise the spiral still further by accomplishing previously unheard-of levels of reuse from system to system.

We believe that product lines will be the predominant software paradigm at the beginning of the new millennium. If the pitfalls are successfully negotiated, the result is, as CelsiusTech Systems has demonstrated, an enviable capacity for delivering very large systems on time and on budget. To date, CelsiusTech has delivered fifty-five systems in its product family. One of the company’s recent quandaries was how many people to assign to

Paradigm (continued on p. 7)

The SPIN Doctor

by Judi Brodman

The Cost of Quality and the ROI

The Question: *How do I find out the cost of quality in software development? Indeed, I don't know much about the cost of quality, and I would like to have some indication of how to determine such a cost.*

The Doctor Answers: For those of you who want to know what the cost of quality (CoQ) is and where it came from, there are two books you should read: *Quality is Free* and *Quality without Tears*, both by Philip B. Crosby. Both books are short, are available in paperback (McGraw Hill), and provide a good background on CoQ.

Cost Categories

Crosby describes CoQ as the extra cost incurred because the product or service wasn't done right the first time. Crosby defines CoQ as a sum of two components: the cost of nonconformance (rework) and the cost of conformance, which is made up of appraisal costs and prevention costs. Nonconformance costs are all those direct and indirect costs associated with reworking a product or service.

Rework Costs. Let's talk about rework for a minute. One of the most costly activities in software development is reworking or redoing what has already been done. For some reason, we have come to accept mountains of rework as part of our every day activities. In some cases, we may not even think of it as rework.

For example, we spend precious schedule time on fixes for code defects when software doesn't perform as expected (functionality or error?) or on redesign of a user interface because the customer expects something different from what we provide (insufficient/incorrect customer requirements?).

Appraisal and Prevention Costs. The cost of conformance includes appraisal costs and prevention costs. Appraisal costs are those associated with evaluating or testing the product or service to determine if it is faulty. Prevention costs, on the other hand, are those derived from the proactive steps one

takes to reduce or eliminate rework. Therefore, appraisal costs are assigned to software quality assurance activities, activities associated with multiple levels of testing (unit, component, system), while prevention costs are assigned to activities such as conducting inspections and/or walkthroughs and the activities associated with the software process improvement program.

Prevention Costs. Raytheon added another cost category: performance costs. These simply include all the costs that are absolutely necessary in developing the software product even in an error-free environment. In other words, this category is the cost of doing it right the first time. With this addition, all project software costs can be allocated to one of the four cost categories.

Raytheon's Experience

Since Raytheon was the first company to shine a spotlight on CoQ as a method for calculating return on investment (ROI) from software process improvement perspective, I contacted Ray Dion to help me answer this question and share Raytheon's experience.

As Dion explained, Raytheon found it difficult to break these four cost categories (rework, appraisal, prevention, and performance) into subcategories, define the subcategories, and map project activities and costs to them. The work breakdown structure (WBS) used on the software projects did not map well to the CoQ categories, and the company's definitions of each subcategory, which were rather brief for reasons of simplicity, were subject to misinterpretation.

"We addressed the first problem," Dion said, "by working toward both a short- and a long-term solution. In the short term, we continued to collect project costs using the conventional WBS, and project leads periodically and manually remapped all costs to CoQ subcategories. In the long term, we developed a common WBS to provide as close a mapping to the CoQ as possible. We expected this also to entail a revision

of the cost accounting system and possibly the time card reporting system as well.

Our approach to the second problem was to refine the definitions as we gained experience using them. This literally required five iterations of the initial data-gathering exercise before we obtained a satisfactory level of consistency.

Following are some of our early experiences using the CoQ model.

- Many questions arose about how to allocate costs to subcategories. There was quite a variation in the methods used to break the actual costs down to the defined cost bin. These were resolved by refining the subcategory definitions and by analyzing and comparing the suballocation algorithms used by the six project leaders.
- Initially it was necessary to have the project leader instead of an administrator generate the data because the project leader possessed first-hand knowledge of project particulars, as well as good engineering judgment.
- The data showed that the average cost of rework decreased following the start of the process improvement initiative. In the two years before the initiative, rework costs averaged about 41 percent of project costs. In the two years following, that value had dropped to about 20 percent, and the trend was continuing downward.

As expected, rework savings were achieved at the expense of a small increase in appraisal or prevention costs. For example, appraisal costs rose when informal reviews were replaced by formal inspections, and prevention costs rose when inspection training was instituted. Also, rework costs associated with fixing defects found during design rose from about 0.75 percent to about 2 percent of project cost. Those associated with fixing defects found during coding rose from about 2.5 percent to about 4 percent of project cost.

The major reduction in rework costs was that associated with fixing source code problems found during integration, which dropped to about 20 percent of its original value. The second largest contributor to rework reduction was the cost of retesting, which decreased to about half its initial value. This clearly indicates that the additional costs of performing

COQ (continued on p. 5)

COQ (continued from p. 4)

formal inspections and the training that must precede it are justified on the basis that finding problems earlier in the process results in a more efficient integration.

- CoQ analysis was not inexpensive, costing about \$25K in overhead. Raytheon repeated the CoQ exercise about a year later and added the process to the normal senior management process reviews on a semiannual basis.”

Lessons Raytheon Learned

Data Gathering More Routine. After three years (by 1993), the data gathering exercise had become more routine. Although the full analysis was being made semiannually, some department managers were requiring that their project leaders provide CoQ data along with their normal monthly tracking data.

Insufficient as a Single Measure of SPI. The CoQ approach would not be sufficient as the single means of measuring the results of software process improvements (SPI). Data were collected from individual projects on their productivity in terms of equivalent delivered source instructions (EDSI) per man-month of development effort. They combined the data from all projects using a weighting function, and the results showed the average productivity was, in fact, increasing as a function of time, meaning that jobs were costing less.

Using Improved Productivity Figures. Management needed to be reassured that the improved productivity figures were being used to bid new jobs. This issue was addressed by collecting data on a project's budgeted (predicted) cost and its actual cost at completion (CAC). This cost performance index ratio (CAC/Budget) for each project was then used to compute the monthly weighted average (again using the same approach as the CoQ) to yield a plot of this time-variant measure. The results were encouraging, showing that the cost performance index changed dramatically, improving from an overrun range of about 20 percent before the start of the initiative to a range of 1 to 2 percent by late 1992 and early 1993.

Transitioning to a Common WBS. More emphasis should have been placed on transitioning to a common WBS geared to the collection of CoQ subcategories. Dion said that the initial approach developed

A Closer Look at Crosby's Definitions And Raytheon's CoQ Experience

by Dan Houston

Crosby's definition, as quoted in the SPIN Doctor column, seems quite adequate for nonconformance costs, but I think it is too restrictive for conformance costs. Many of these costs are incurred to assure quality, even if it's already there. Thus, saying that it only includes costs because the product/service wasn't right the first time doesn't seem to allow for the cost of preventive activities. I find that in the literature some start out with a restrictive definition of cost of quality (CoQ), then broaden it when they start including prevention activities.

I think it's best to start with a definition that is broad enough to include prevention; that is my attempt when I suggest that CoQ includes such activities performed primarily out of concern for quality vis a vis functionality, schedule, and budget.

I like Ray Dion's description of the costs of conformance for software. I find that categorizing these costs between prevention and appraisal varies somewhat with different authors for the reasons just mentioned. Regarding software quality assurance activities, I would add a distinction used by the American Society for Quality (ASQ): it considers quality administration (such as quality program planning, reporting, and education) as preventive and audits of product and production processes as appraisal.

Regarding the performance category added by Raytheon, no concrete examples of its activities are offered. It is added so that all project costs can be allocated to a quality cost category. This is a departure from the ASQ approach, however, because it would mean that total CoQ is equal to total project costs. Taking this total-costs view of CoQ defeats its original purpose of showing the contributions of quality improvement to product profitability.

for calculating CoQ is still being used today.

Conclusion

According to Dion, the CoQ analysis used by Raytheon's Equipment Division is a viable mechanism for measuring the overall effect of software improvement. By using it to monitor and reduce software CoQ, productivity and predictability can

Quality costs represent the expenses when an organization has the opportunity to reduce spending by improving its quality-inducing processes. This is especially true today in software development; software quality is the most difficult of the four project dimensions to define and is often sacrificed to budget, schedule, and functionality. Measuring and monitoring CoQ helps to define product quality but only if it's not watered down by including all other development costs.

There is one other disadvantage of equating CoQ and total project costs. Cost of software quality is often shown as a percentage of development costs. This allows a company to see the financial benefits of software process improvements across projects and over time. This benefit disappears, however, if all project costs are accounted as quality costs.

Raytheon's difficulties with misinterpretation of quality categories shows the importance of an organization having a good CoQ chart of accounts when it starts measuring CoQ. This is emphasized in the lessons the company learned. Although most of the categories in a chart may not be used at the outset, a good chart provides the definition required as quality activities mature and accounting questions arise.

Raytheon said that CoQ is not sufficient as the only method of measuring results of process improvements. I'm sure that is true, but I am coming to the conclusion that it is the easiest and most cost-effective way for a level 1 or 2 organization to begin seeing the results of software process improvement because it is straightforward and does not require a large effort to start. As interest in CoQ builds, more effort can be invested in it, as Raytheon's experience shows.

be increased. It also can be used to isolate software scrap and drive it to zero. Applying the CoQ approach also provides early feedback and can help improve an organization's standard process.

Send your comments and questions to the SPIN Doctor at brodman@tiac.net

SQA and the Dodo Bird: the Modern Role of the Software Quality Engineer

by Herb Krasner and Linda Shafer



One of Dr. Deming's Total Quality Management (TQM) principles urges us to cease depending on testing, back-end inspections, and process audits to achieve quality.

It's too late to do much about defect correction at the end of the development cycle—the pressure is on to ship the product. Introduction of quality activities early and often is the only way to achieve truly built-in quality.

Modern software developers would do well to heed Dr. Deming's advice. Doing so, however, signals the demise of the old-fashioned software quality assurance (SQA) specialist and the birth of the software quality engineer (SQE). The SQE augments the development team and ensures that quality activities are integrated from the very beginning of the life cycle.

An Endangered Species

The SQA specialist is an endangered species for several reasons.

- Testing alone can never prove that a software product is of high quality. (Unfortunately, many organizations still think that testing and SQA are the same thing.)
- Inspections of code occur much too late to allow for thoroughly tested corrections.
- Process audits of an uncontrolled process yield little value—there is little to audit against!

Many organizations are rethinking the need for an SQA function at all. The SQE is better equipped as a catalyst for quality improvement, which begins with a controlled process. This is evidenced by the many companies that have been working at achieving higher software process maturity for more than 10 years and that have reaped the associated software quality (and productivity) benefits.

Characteristics of a Mature Organization

In a highly mature organization, company policy requires all business units to use approved and documented processes to guide the development and maintenance of software. Such policies support the goals of ensuring high quality and meeting customer expectations.

In a mature organization, a defined and approved process demonstrates that software:

1. is developed in support of documented requirements and conformance to these requirements is validated (e.g., through reviews, inspections, and formally developed acceptance tests);
2. is developed using a phased process where entry criteria are defined and exit criteria require structured reviews and approvals (criteria that bolster all life cycles and do not conflict with iterative prototyping or concurrent engineering life cycles);
3. is developed and maintained under documented plans for configuration management and change control, including installation and customer configuration;
4. is developed using formalized analysis and design methods;
5. is developed using current standards and conventions for requirements, design, construction, documentation, and testing;
6. has appropriate documentation for each development phase and interim deliveries of the product;
7. undergoes system/acceptance testing by individuals or organizations not directly involved in the design or implementation of the product being tested;
8. undergoes controlled testing in a customer representative system before general product release; and
9. demonstrates, through metric analysis, the achievement of specific software quality goals.

The Modern SQE

The modern SQE nurtures this process with best practices for quality and acts as a customer advocate in software quality matters. The SQE is responsible for teaching software quality principles and practices and for leading the organization in the practical application of the teachings.

A number of additional activities fall within the SQE's realm. The SQE, for example, helps project management select and tailor a software life cycle and approach to achieving high-quality software on each project. The SQE jointly develops, with project management, a software quality assurance plan (SQAP) for each specific

software-intensive project and audits the project for compliance with the SQAP, the software development plan (SDP), and other existing standards and guidelines.

The SQE participates in the assessment and metric tracking of internal deliverables, as well as in the measurement, analysis, and feedback of overall cost of software quality, levels of software quality, and customer satisfaction. In addition, this engineer witnesses or audits tests and attends selected peer reviews or inspections to assure appropriate software verification and validation. In some cases, the SQE coordinates and/or performs the testing, depending upon your

SQA (continued on p.7)

Software Quality Matters

Editor: Cinda L. Cyrus

© 1998 by The University of Texas at Austin. All rights reserved

Printed in the United States of America

Editorial Board Members

Al Dale	Cinda Cyrus
Jack Odom	Herb Krasner
Joanne Click	

Contributors to this Issue

Judy Brodman	Cinda Cyrus
Candy Walser-Berry	Paul Ehrler
Debbie Caples	Gale Hathcock
Paul Clements	Don Houston
Joanne Click	Herb Krasner
Gary Cobb	Linda Shafer

Software Quality Matters is published quarterly by the Software Quality Institute at The University of Texas at Austin College of Engineering, in cooperation with the Austin Software Council and the Austin Software Process Improvement Network.

Brand names, company names, trademarks, or other identifying symbols appearing in text and illustrations are used for educational purposes only and do not constitute an endorsement by the authors, editor, or publisher.

For additional information, contact —

Software Quality Institute
The University of Texas at Austin
PRC MER Mailcode R9800
Austin TX 78712-1080
Phone: (512) 471-4875
or: 1-800-687-8012
Fax: (512) 471-4824
E-mail: info@sqi.utexas.edu
WWW: <http://www.utexas.edu/coe/sqi>

SQA (continued from p. 6)

organization's definition of the SQE and tester roles.

Providing appropriate visibility in situations of nonconformance is another important activity for the SQE, as is participating in change control and associated processes. The SQE is also responsible for assuring that quality phase entry- and exit-gate criteria exist and are used to facilitate transition to the next phase of the project's process and that phase and finally project post-mortem and preview analyses are done.

As part of the continuous improvement process, the SQE also is the ideal person to coach and/or train software development team members in the use of best practices for high-quality achievement.

Beyond the project-specific role, the modern SQE also serves as a cost-of-software-quality reduction champion, helps foster a quality culture, leads training efforts in software-quality topics, and participates in continuous software process improvement activities.

The SQE's Knowledge Base

In order to perform the role effectively, the modern SQE practitioner must have practical working knowledge of the following key subjects.

- TQM techniques and software quality principles
- Software quality standards and ethics
- Software quality planning and management techniques
- Software processes and maturation approaches
- Metrics, measurement, and analytical methods
- Inspections, testing, and verification and validation methods
- Product assurance techniques
- Process audit techniques
- Software configuration and change management techniques
- Basic software project and quality management practices

Pursuing the American Society for Quality's (ASQ) Certified Software Quality Engineer—a credential highly recommended for modern SQE professionals—is a good way to learn more about these subjects.

If members of your software quality staff have this knowledge and assume the responsibilities outlined in this article, then your customers are right to expect high-quality software systems from your organization.

SQE Seminars Offer Opportunities To Learn New Techniques, Skills



SQE is happy to announce the following spring courses for software professionals. For more details see www.utexas.edu/coe/sqi.

Practical In-Process Inspections—*Geree Streun, instructor*. This two-day course offered March 9–10 examines the advanced technique known as in-process inspection. Through a series of small inspections during development, the technique is designed to find defects in each and every software development process artifact (e.g., project plans, requirements documentation, design documentation, code, and test documentation).

Software Configuration Management—*Cecil E. Martin, instructor*. This three-day seminar, which is offered March 16–18 is designed to help an organization accurately implement a framework for information systems centered around baselining and controlling automated systems that are critical to business missions.

Software Testing—*Gary W. Cobb, instructor*. Designed for software system engineers, this three-day course, which meets March 30–April 1, covers the role of software testing throughout the life cycle of the typical software development process.

Software Risk Management—*Joel Gittleman, instructor*. This two-day seminar, offered April 6–7, will survey the processes and practices used for software risk management to introduce students to the key components of a risk-management process.

Software Capability Maturity Model—*Joel Gittleman, instructor*. This one-day seminar, offered April 14, provides an overview of the SEI Capability Maturity Model (CMM), describes its purpose and structure, and examines its use in software process improvement.

Software Requirements Engineering and Management Processes—*Cecil E. Martin, instructor*. Poorly stated and poorly understood problem domain requirements lead to 56 percent of software defects and more than 82 percent of the cost for debugging and fixing software. This three-day course, which is offered April 20–22, addresses the most difficult and neglected software engineer activities: analyzing the problem domain and formally documenting requirements.

Multimedia Systems Design—*Gary Cobb, instructor*. In this two-day seminar, offered April 30–May 1, a number of multimedia applications will be demonstrated, a variety of authoring tools will be discussed and demonstrated, and a number of issues surrounding the integration of multimedia content in a distributed computing environment will be raised and examined.

Software Architecture in Practice—*Paul Clements and Rick Kazman, instructors*. This two-day seminar, which is offered May 18–19, provides an in-depth introduction to the concepts and practices associated with software architecture.

Software architecture is the keystone of successful product development. It is the first and best opportunity to ensure that the system meets its quality goals; it is the basis for organizing the development effort; and it is the manifestation of the earliest design decisions.

Additional Information

For more information about the courses, contact—

Software Quality Institute

Phone: (512) 471-4922

E-mail: leti@sqi.utexas.edu

WWW: <http://www.utexas.edu/coe/sqi>

Paradigm (continued from p. 3) integration testing for these 1.5-million SLOC systems. It believed that one person could do the job, but, worrying that the engineer might become lonely, the company often adds a second person. Would that we all had quandaries like that.

Additional Information

For more information about SEI's Product Line Systems Program and its technology initiatives, visit our web page at http://www.sei.cmu.edu/technology/product_line_systems. You can download the full report detailing the CelsiusTech product line case study and other product-line-related reports. You can also contact Linda Northrop, Product Line Systems Program Manager, at lmn@sei.cmu.edu

About the author. *Paul Clements is a senior member of the technical staff at Carnegie Mellon University's Software Engineering Institute. He is co-author of Software Architecture in Practice (Addison-Wesley, 1997), along with Len Bass and Rick Kazman of the SEI.*

SQI Calendar—Meetings and Conferences of Interest

March 9-10, 1998

SQI Seminar
Practical In-Process Inspections
Pickle Research Campus, Austin, TX

March 13, 1998

SQI Seminar—First meeting of *Personal Software Process Training*
Pickle Research Campus, Austin, TX

March 16-18, 1998

SQI Seminar
Software Configuration Management
Pickle Research Campus, Austin, TX

March 17, 1998

SQI Seminar
Software Size & Cost Estimation
Pickle Research Campus, Austin, TX

March 19, 1998

A-SPIN monthly meeting
6:30 to 9:30 p.m.
Pickle Research Campus, Austin, TX

March 30-April 1, 1998

SQI Seminar
Software Testing
Pickle Research Campus, Austin, TX

April 2, 1998

Second Annual SQI Symposium on Software Quality
Pickle Research Campus, Austin, TX

April 6-7, 1998

SQI Seminar
Software Risk Management
Pickle Research Campus, Austin, TX

April 14, 1998

SQI Seminar
Software Capability Maturity Model
Pickle Research Campus, Austin, TX

April 16, 1998

A-SPIN monthly meeting
6:30 to 9:30 p.m.
Pickle Research Campus, Austin, TX

April 20-22, 1998

SQI Seminar
Software Requirements Engineering and Management Processes and Methods
Pickle Research Campus, Austin, TX

April 30-May 1, 1998

SQI Seminar
Multimedia Systems Design
Pickle Research Campus, Austin, TX

May 18-19, 1998

SQI Seminar
Software Architecture in Practice
Pickle Research Campus, Austin, TX

May 28-29, 1998

SQI Seminar
Verification and Validation Processes and Methods
Pickle Research Campus, Austin, TX

June 28-July 1, 1998

IEEE International Conference on Multimedia
Pickle Research Campus, Austin, TX

July 8-9, 1998

SQI Seminar
Software Project Planning and Tracking
Pickle Research Campus, Austin, TX

August 10-11, 1998

SQI Seminar
Software Risk Management
Pickle Research Campus, Austin, TX

Symposium (continued from p. 2)

of the cost of software quality (CoSQ) can be used as a means of providing justification to management and feedback to staff, for investments in process improvement, to get management's attention, and to provide a measurement baseline for determining how software quality improvement efforts are progressing.

Krasner is president of Krasner Consulting and chair of SQI's Advisory Group.

Using Software Failure Analysis Data

Analyzing the causes of software defects, Grady believes, can be used to motivate change. In his presentation, he will show how this analysis can be used to gain insight into significant process weaknesses, link these weaknesses back to business needs, and use such analyses to motivate and track improvements that build competitive advantages.

Grady is software metrics program manager for Hewlett-Packard's Corporate Engineering Software Initiatives.

Politics and Polemics

In describing the creation and implementation of a measurement program in ITT, one of the largest corporate software productivity and quality measurement programs ever attempted, Curtis will discuss the technical and political problems that had to be solved in implementing the corporate-wide program. The problems were agreement on uses of the data, comparability of data definitions, reliability of data, conundrums in data analysis, and turning data into useful corporate information.

Curtis is co-founder and chief scientist of TeraQuest Metrics.

Additional Information

For additional information or to register for this symposium, see SQI's web page at www.utexas.edu/coe/sqi



Software Quality Institute
The University of Texas at Austin
PRC MER R9800
Austin, Texas 78712-1080

Nonprofit Org.
U.S. Postage
PAID
Austin, TX
Permit No. 391