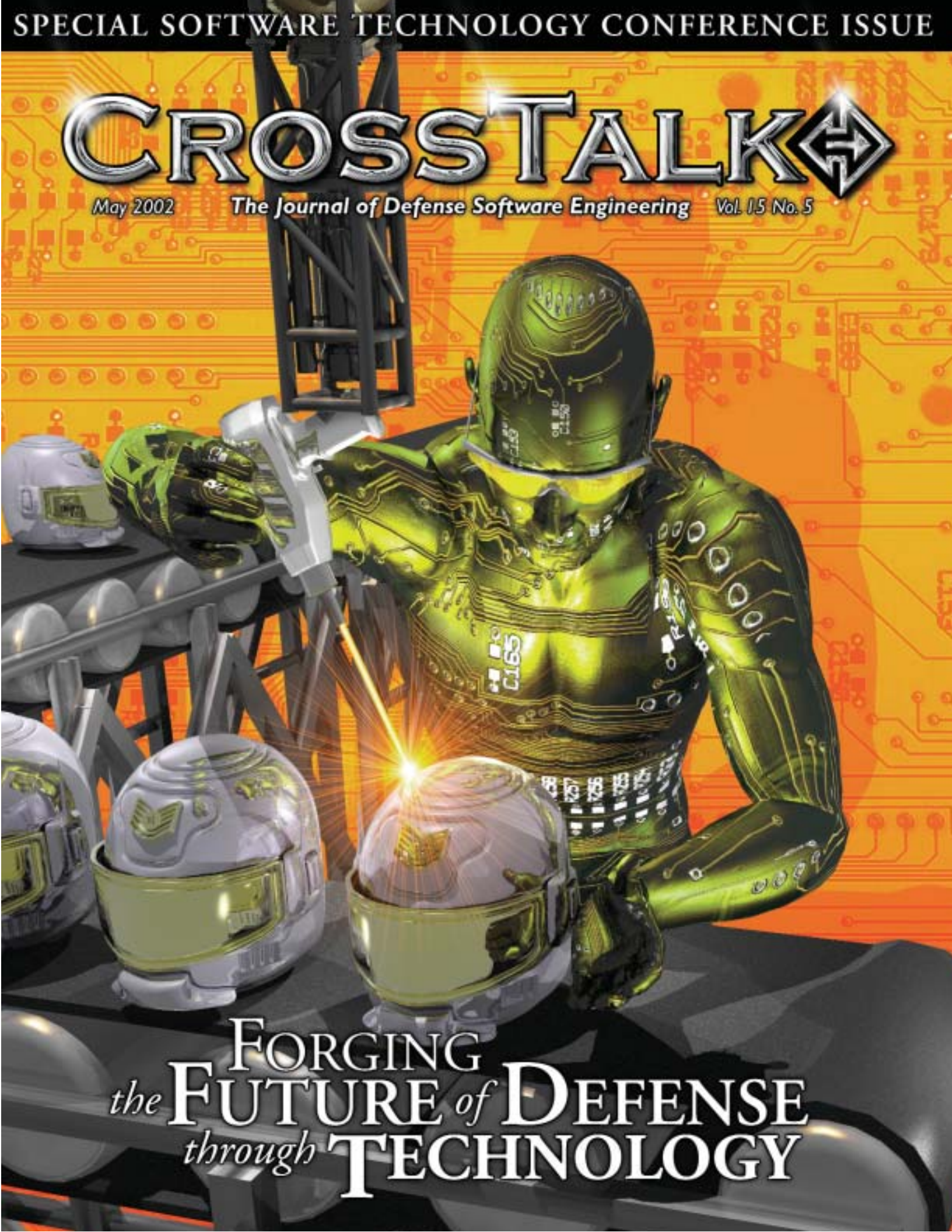
SPECIAL SOFTWARE TECHNOLOGY CONFERENCE ISSUE

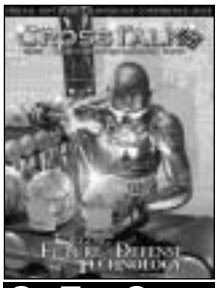
CrossTalk

May 2002 The Journal of Defense Software Engineering Vol. 15 No. 5



Forging the Future of Defense through Technology

Best Practices



ON THE COVER
Cover Design by Kent Bingham.

Software Engineering Technology

Open Forum

Online Article

Departments

CrossTalk

SPONSOR	*Lt. Col. Glenn A. Palmer*
PUBLISHER	*Tracy Stauder*
ASSOCIATE PUBLISHER	*Elizabeth Starrett*
MANAGING EDITOR	*Pamela Bowers*
ASSOCIATE EDITOR	*Julie B. Jenkins*
ARTICLE COORDINATOR	*Nicole Kentta*
CREATIVE SERVICES COORDINATOR	*Janna Kay Jensen*
PHONE	(801) 586-0095
FAX	(801) 777-8069
E-MAIL	crosstalk.staff@hill.af.mil
CROSSTALK ONLINE	www.stsc.hill.af.mil/crosstalk
CRSIP ONLINE	www.crsip.hill.af.mil

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 25.

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

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 Author Guidelines, available at www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf. CROSSTALK does 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: This DoD journal is an authorized publication for members of the Department of Defense. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the government, the Department of Defense, or the Software Technology Support Center. All product names referenced in this issue are trademarks of their companies.

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 Online Services: www.stsc.hill.af.mil
Call (801) 777-7026, e-mail: randy.schreifels@hill.af.mil

Back Issues Available: The STSC sometimes has extra copies of back issues of CROSSTALK available free of charge.

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 to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

What Are the Future Challenges of Software Technology?

As this issue of CrossTalk goes to print, the Software Technology Support Center (STSC) and Utah State University are busy working the final details of the 14th Annual Software Technology Conference 2002 – "Forging the Future of Defense Through Technology." With abstract submittals up dramatically from the past year and a slate of distinguished keynote speakers, we are confident this year's conference will be as strong as ever.

What is the future of software technology? As military departments transform operations, commands define new methods of force employment, and industry develops the next generation of advanced weapons, the prognosis of many software technocrats is unfolding – increased demand for quality, rising importance of information assurance, and new requirements demanding interoperability and exchange of data across systems. These technical challenges must be met while competing for talent, training a new workforce, and developing more mature organizations that can consistently deliver new products on schedule while increasing efficiency. The articles in this month's CrossTalk explore some of these challenges.

First we present the results of a study by Dr. Richard Turner, faculty member of The George Washington University and assistant deputy director for the Software Intensive Systems Office of the Under Secretary of Defense. In *A Study of Best Practice Adoption by Defense Acquisition Programs*, Dr. Turner states that a survey of 14 software centers involving 150 programs indicated that despite demonstrated effectiveness and awareness, only about 25 percent of programs fully adopt any given best practice. He also explores barriers to implementation. The data show that with regard to software development and acquisition practices, leadership must focus on implementation.

How can organizations develop a culture and practices to achieve success? Dr. Barry Boehm, Dr. Daniel Port, and Apurva Jain of the University of Southern California and Dr. Victor Basili of the University of Maryland address this question in part four of a series of articles for CrossTalk. Building on the project-level benefits of Model-Based (system) Architecting and Software Engineering (MBASE), and Schedule as Independent Variable (SAIV) approaches in the previous article, this month's work, *Achieving CMMI Level 5 Improvements with MBASE and the CeBASE Method*, describes how to address the CMMI organization-level process areas, particularly those of achieving continuous improvement.

The need for cooperation and data sharing among intelligence, military, and law enforcement organizations is the backdrop for *U.S. Defense Department Requirements for Information Security* by Kevin J. Fitzgerald of Oracle Corporation. Information security and multi-level security requirements are becoming primary considerations for the design of any major system (weapon or information). Fitzgerald outlines the basic requirements and information security terminology and makes a case for secure, independently evaluated solutions that incorporate security into the entire computing infrastructure.

Following this, Dr. Linda H. Rosenberg, Goddard Space Flight Center, NASA, in her article, *What is Software Quality Assurance?* defines the link between system safety and mission success to software quality assurance.

As the size of software products grows, effective test automation becomes necessary. In our next article, *Surviving the Top 10 Challenges of Software Test Automation*, Randall W. Rice discusses common problems with test automation and describes strategies for improvement. Making test automation an integral part of the organization and engineering process is necessary to gain its benefits.

Lastly, *Information Security System Rating and Ranking* by Dr. Rayford B. Vaughn Jr., Ambareen Sira, and Dr. David A. Dampier all of Mississippi State University summarizes information gathered from a joint workshop conducted with both government and commercial sector engineers. The result was a characterization of information security metrics, and a case that processes, procedures, tools, and people all interact to produce assurance in systems. An effective set of measures must incorporate all these areas.

These topics – best practices, implementing organizational change, metrics, quality assurance, test automation, and information security – are but a few of those to be presented and displayed at this year's Software Technology Conference, April 29-May 2, 2002. We hope to see you there as we all share ideas that will carry us into the future of software technology.

Glenn A. Palmer

Lt. Col. Glenn A. Palmer
Director, Computer Resources Support Improvement Program

A Study of Best Practice Adoption by Defense Acquisition Programs

Dr. Richard Turner
The George Washington University

Tuesday, 30 April 2002
Track 1: 3:00 - 3:40
Ballroom A

Best practices are often recommended as a way to improve the success and quality of software-intensive system acquisitions within the Department of Defense. However, one of the primary questions raised by this recommendation is, "To what degree have existing projects, in fact, adopted the best practices?" While the assumption is that most programs are not using best practices, there are few specific studies available. This article presents the results of a study that measured best practice adoption in defense acquisitions.

The United States Department of Defense (DoD) spends an estimated $20 billion a year on software to support its infrastructure; operate its weapons systems; and provide command, control, communications, computing, intelligence, surveillance, and reconnaissance functions. DoD acquires the large majority of this software from contractor sources.

There have been significant cost overruns and schedule delays experienced in DoD software-intensive system acquisitions, resulting in numerous audits and evaluations of acquisition programs by independent government and industry organizations. Such evaluations have consistently indicated that programs are at risk partially because of failure to implement best practices. The evaluations have recommended the implementation of a variety of practices to improve performance [1-8].

A study by Anderson and Rebentisch [9] of 23 military programs found that practice implementation for eight recommended commercial practices[1] ranged from 17 percent to 83 percent with essentially half of the practices implemented under 40 percent of the time and half over 50 percent of the time.

This article reports on research conducted to estimate how broadly acquisition best practices are implemented within DoD. The study involved developing and conducting a survey to establish the implementation and perceived effectiveness of a set of best practices.

Study Methods

The first critical issue was deciding whom to survey. It was desirable to obtain as wide a sample as possible with the least amount of interference in the acquisition program activities. For this reason, it was determined that the most effective way to access a wide variety of projects was to contact the various military software centers that provide software expertise to the programs. These centers act as intramilitary consultants or centers of excellence to provide expert resources to the acquisition program offices. Their personnel are in a position to provide informed judgments without any political bias from program loyalty.

> *"... evaluations have consistently indicated that programs are at risk partially because of failure to implement best practices."*

The second critical decision was selecting the best practices for evaluating adoption. Since one of the objectives of this research was to determine how to support the implementation of best practices, it was decided to use the most widely known and oldest set of practices as the baseline for the adoption study. This would maximize the odds that program managers would have heard of the practices and that the acquisition personnel would have encountered them in use. Therefore, the original nine Airlie practices[2] were used in the survey.

The following definitions of the Airlie practices are derived from the Software Program Managers Network (SPMN) materials [10, 11]. Copies of the SPMN material defining the practices were included in the materials sent to the survey participants. They are as follows:

1. Formal Risk Management. A formal risk management process requires acceptance of risk as a major consideration; commitment of program resources to managing risk; and use of planned, documented methods for identifying, monitoring, and managing risks.
2. Agreement on Interfaces. A baseline interface specification must be established and agreed to by all stakeholders before implementation activities begin. A separate software specification must be developed with explicit and complete interface information. This is particularly critical with human/machine interfaces and where system interoperability is a requirement.
3. Formal Inspections. Inspections of all acquisition and development documentation should be conducted according to planned, documented processes and the results placed under configuration control, tracked, and resolved.
4. Metrics-Based Scheduling and Management. Statistical quality control of costs and schedules should be maintained. Reasonable cost and schedule projections should be made before program start, and specific measurement processes should be put in place early in the program and rigorously followed. Measurement results should figure prominently in program reviews and management decisions.
5. Binary Quality Gates at the Inch-Pebble Level. Status should be tracked through binary completion of relatively small tasks. Activities are either incomplete or complete. This is to prevent the "80-percent-complete" syndrome where the estimated completion figure is reported without particular rigor.
6. Program-Wide Visibility of Progress vs. Plan. Core indicators of project health and performance should be readily available to all project participants. Anonymous feedback channels

should be provided to enable bad news to be propagated up and down the project hierarchy without fear of reprisal for truth telling.

7. Defect Tracking Against Quality Targets. Defects should be tracked according to a planned, documented process; measured against established targets; and systematically tracked through removal or resolution.
8. Configuration Management. A planned, documented process is followed to identify, document, monitor, evaluate, control, and approve changes made during the system life cycle to any system-related artifact that is shared by more than one individual or organization.
9. People-Aware Management Accountability. Management should treat personnel as their principle resource by staffing qualified people, encouraging continuous improvement, and fostering an environment conducive to low voluntary personnel turnover.

Survey Instrument

In developing the instrument, we found it useful to think of adoption as having two components: awareness and implementation. Awareness, as defined by Hilburn [12], represents a level of individual knowledge about the practice that includes the following:

- Understanding of the existence and context of the practice within the context of software acquisition.
- A general, informal explanation of the practice.
- Identification of references (human/written) that provides greater depth of knowledge about the practice.

The other component, implementation, requires that the organization put into place the requisite infrastructure, training, resources, and policy to effectively utilize the practice in doing business.

The adoption survey instrument was designed to provide data that addressed both awareness and adoption. It collected data in the following areas:

1. The number of programs supported by the respondent to establish the overall program sample.
2. The size of those programs as designated by Acquisition Category (ACAT): ACAT I, ACAT II, ACAT III, or Other.[3]
3. The quality of practice adoption for each program as measured by the compliance with the practice definition in the Airlie material. This was captured by having each participant differentiate between full compliance and partial compliance for each of the projects. Partial compliance would be a surrogate for *awareness*, while full compliance would represent *implementation*. Obviously, this is not a perfect surrogate. However, since partial compliance does, in fact, capture awareness, the worst error would be that of underestimating awareness. It was decided that such an error could be dealt with by considering it in the analysis. The data were gathered for each Airlie practice as the following:
   - The number of programs in each size category that fully implemented the practice.
   - The number of programs in each size category that implemented some facet of the practice.
4. An evaluation of the perceived overall effectiveness of the practice as observed by the center personnel measured on a five-point scale: Highly Effective (5), Very Effective (4), Moderately Effective (3), Somewhat Effective (2), Negligibly Effective (1). This scale was chosen to reflect that as best practices, the practices were, at worse, ineffective.

Service	ACAT I	ACAT II	ACAT III	Other	Total
Air Force	1	0	16	62	**79**
Army	9	11	13	6	**39**
Navy	2	6	7	17	**32**
Total	**12**	**17**	**36**	**85**	**150**

Table 1: *Programs Reported by Service and ACAT Designation*

ACAT I	ACAT II	ACAT III	Other
8%	11%	24%	57%

Table 2: *Percentage of Programs Represented by ACAT Designation*

Air Force	Army	Navy
53%	26%	21%

Table 3: *Percentage of Programs Represented by Service*

Results

Of the 14 centers asked to participate, seven responded to the data call. Six of the seven provided the requested data:

1. Army TACOM TARDEC
2. Army CECOM
3. Navy NAVSEA
4. Navy NUWC
5. Navy NAVAIR
6. Air Force ESC

The seventh respondent, Air Force ASC, provided narrative comments only. The responses covered 150 software acquisition programs broken out as shown in Tables 1 through 3.

Airlie Practice Adoption Data

The responses to the survey resulted in 1,350 possible program-practice pairs (150 programs times nine practices) where a particular Airlie practice could be adopted by a particular program. Table 4 shows the summarized results of the survey when calculated against this full complement of program-practice pairs. The terms *partial* and *full* refer to whether the respondent indicated that the program partially or fully implemented the practice.

Table 5 presents the adoption data by practice. The percentages represent the

Table 4: *Overall Results of Adoption Study (Percent of Possible Program/Practice Pairs)*

Overall	ACAT I	ACAT II	ACAT III	Other	Total
Partial	67%	52%	38%	69%	**59%**
Full	13%	29%	31%	24%	**25%**
Total	**80%**	**80%**	**69%**	**93%**	**84%**
Air Force	**ACAT I**	**ACAT II**	**ACAT III**	**Other**	**Total**
Partial	44%	None	56%	78%	**73%**
Full	56%	None	12%	21%	**20%**
Total	**100%**	**None**	**68%**	**99%**	**93%**
Army	**ACAT I**	**ACAT II**	**ACAT III**	**Other**	**Total**
Partial	64%	45%	21%	13%	**36%**
Full	9%	24%	50%	67%	**36%**
Total	**73%**	**70%**	**70%**	**80%**	**72%**
Navy	**ACAT I**	**ACAT II**	**ACAT III**	**Other**	**Total**
Partial	89%	63%	29%	54%	**52%**
Full	11%	37%	40%	18%	**26%**
Total	**100%**	**100%**	**68%**	**72%**	**78%**
Note: Numbers may not sum due to rounding					

Best Practice	ACAT I		ACAT II		ACAT III		Other	
	Partial	Full	Partial	Full	Partial	Full	Partial	Full
Formal risk management	92%	8%	41%	29%	39%	31%	72%	19%
Agreement on interfaces	75%	25%	41%	59%	47%	36%	67%	29%
Formal inspections	50%	8%	71%	0%	36%	28%	65%	20%
Metric-based scheduling and management	83%	8%	59%	12%	44%	22%	69%	19%
Binary quality gates at the inch-pebble level	33%	0%	65%	6%	14%	22%	71%	16%
Program-wide visibility of progress vs. plan	83%	8%	47%	53%	47%	36%	72%	26%
Defect tracking against quality targets	25%	17%	29%	41%	31%	36%	68%	31%
Configuration management	67%	33%	47%	53%	42%	42%	66%	34%
People-aware management accountability	92%	8%	65%	6%	42%	25%	68%	21%

Table 5: *Adoption Results by Practice*

Best Practice	Effectiveness*					Effectiveness Score*
	HE	VE	ME	SE	NE	
Formal risk management	14%	**50%**	29%	7%	0%	3.71
Agreement on interfaces	36%	**43%**	21%	0%	0%	4.14
Formal inspections	21%	**36%**	14%	21%	7%	3.43
Metric-based scheduling and management	15%	38%	**46%**	0%	0%	3.69
Binary quality gates at the inch-pebble level	18%	18%	**36%**	27%	0%	3.27
Program-wide visibility of progress vs. plan	15%	**69%**	8%	8%	0%	3.92
Defect tracking against quality targets	23%	**38%**	**38%**	0%	0%	3.85
Configuration management	33%	**47%**	20%	0%	0%	4.13
People-aware management accountability	14%	21%	**43%**	21%	0%	3.29

*5=Highly Effective (HE), 4=Very Effective (VE), 3=Moderately Effective (ME), 2=Somewhat Effective (SE), 1=Negligibly Effective (NE).

Table 6: *Effectiveness Results by Practice*

number of projects reporting the particular value divided by the total number in the ACAT designation category.

Airlie Practice Effectiveness Data

Table 6 presents the effectiveness ratings by practice as percentages of respondents rating the practice in the particular categories from Highly Effective (HE), Very Effective (VE), Moderately Effective (ME), Somewhat Effective (SE), through Negligibly Effective (NE). The effectiveness value is the mean of the scores received using five as the value for HE, four for VE, three for ME, two for SE, and one for NE. Table 7 shows the overall results with the practices ranked by effectiveness.

Common barriers reported were as follows:

- Lack of Resources. In general, program offices lack sufficient software-educated staff to implement many of the practices. Some lack sufficient core staff and are too busy fighting everyday fires to even consider best practices.
- Inadequate Contracts. Some contracts do not allow the flexibility needed to implement practices that require developer support without costly time- and energy-consuming modifications.
- Data Accuracy, Availability, and Latency. Practices often depend on accurate, timely data that are simply not available to many DoD programs. The lack or unavailability of valid, useful historical data often stymies estimation practices.
- Management Awareness and Commitment. Several comments were made about managers who did not understand that there was a problem or who did not want to spend the resources on a practice that might actually add risk rather than reduce it. Management must be willing to expend resources and perhaps political capital to institute practices.
- Lack of Credible Evidence. Comments were received concerning the need to prove that the benefits of practices were worth the cost of practice implementation.

Table 7: *Overall Adoption and Effectiveness (Practices in Order of Effectiveness Ranking)*

Overall	P+F*	F* Only	Effectiveness**
Agreement on interfaces	94%	34%	4.14
Configuration management	96%	38%	4.13
Programwide visibility of progress vs. plan	94%	30%	3.92
Defect tracking against quality targets	83%	32%	3.85
Formal risk management	84%	22%	3.71
Metric-based scheduling and management	81%	18%	3.69
Formal inspections	76%	19%	3.43
People-aware management accountability	83%	19%	3.29
Binary quality gates at the inch-pebble level	69%	15%	3.27

*P+F = Partial and full implementations combined; F Only = Full implementations only.
**5=Highly Effective, 4=Very Effective, 3=Moderately Effective, 2=Somewhat Effective, 1=Negligibly Effective.

Analysis

The first observation is that the practices are widely recognized across the programs as indicated by the high percentage (84 percent) of either partial or full implementations. Any particular practice was implemented in some form for at least 69 percent of the projects reporting. Three of the practices were implemented in around 95 percent of the projects. However, when only considering full implementations, those figures drop dramatically to an average of 25 percent across all programs with the lowest rate for a specific practice adoption of 15 percent and the highest rate of 38 percent.

Across services, adoption rates were generally consistent. The Army had a generally higher percentage of full implementation. Figure 1 illustrates the relationships between the services.

The practices were evaluated as relatively effective, with the majority of responses falling in the *moderately effective* to *very effective* range. The practice considered least effective (binary quality gates) still had a mean score of 3.27, placing it in the moderately effective category.

If we look at the adoption and effectiveness data together, there is some correlation between the perceived effectiveness of a practice and its adoption. This seems logical, since in budget- and schedule-constrained programs, the practices with the highest effectiveness would seem more likely to be implemented.

Conclusions

As stated earlier, we have used partial compliance as a surrogate that implies awareness and full compliance to imply implementation. Therefore, the primary finding from the adoption research is that despite the widespread awareness of the best practices (the average of programs implementing practices was 85 percent, which as we

previously noted, is probably an understatement of true awareness), there is very little actual implementation (average 25 percent). If we assume that the practice must be fully implemented to gain substantial benefit, little value is being realized.

In general, full implementation is not required; however, when coupled with the environment of defense acquisition, full (and possibly formal) implementation is the only way that a practice can expect to maintain focus and resources long enough to achieve benefits. This is particularly true of practices that have longer benefit latency.

Judging by the effectiveness ratings, the Airlie practices have stood the test of time and represent valid best practices. Within the Airlie practices, configuration management, agreement on interfaces, and risk management are essentially fundamental project management activities.

As one respondent pointed out, "A number of the things promoted by the SPMN are simply established good practices that were known and practiced before the Airlie group documented them as best practices." That said, there are still many programs that do not implement these practices effectively and so should be reminded of their importance.

The research as conducted describes an environment where managers are aware of the benefits of acquisition practices, but they do not implement them. Either the barriers that prevent full implementation are sufficiently high to deter action, or the program managers simply choose not to implement the practices. The research supported both of these possibilities.

To reap the benefits of the Airlie practices, or any best practices or other acquisition technology, the software-intensive system acquisition environment needs to be changed. The Software Intensive Systems (SIS) office within the Acquisition Resources and Analysis Directorate of the Under Secretary of Defense for Acquisition, Technology, and Logistics is working to improve policy, transition new acquisition technology into programs, coordinate independent expert program reviews, gather empirical data on best practices, and support broader software-related education across the acquisition workforce.

SIS has completed additional research in the area of best practices in the last year. I will be writing another article that will describe the consolidation of more than 100 published practices into 32 candidate practices. Those practices were evaluated for effectiveness by a panel of experts, and an analysis of their impact on software-intensive, system-acquisition risk areas was performed. Further research on better ways of describing practices in a way more suitable to selection and evaluation by acquisition personnel is in the final stages. The results will be briefed in the SIS track during the 2002 Software Technology Conference. The best practice research will be combined with other SIS efforts, including the TriService Assessment Initiative and the CeBASE Experience Factory pilots, to support better decision making and improved processes in software-intensive system acquisitions across DoD. Further work on documenting and disseminating best practices is being performed in collaboration with the Data and Analysis Center for Software in Rome, N.Y.◆

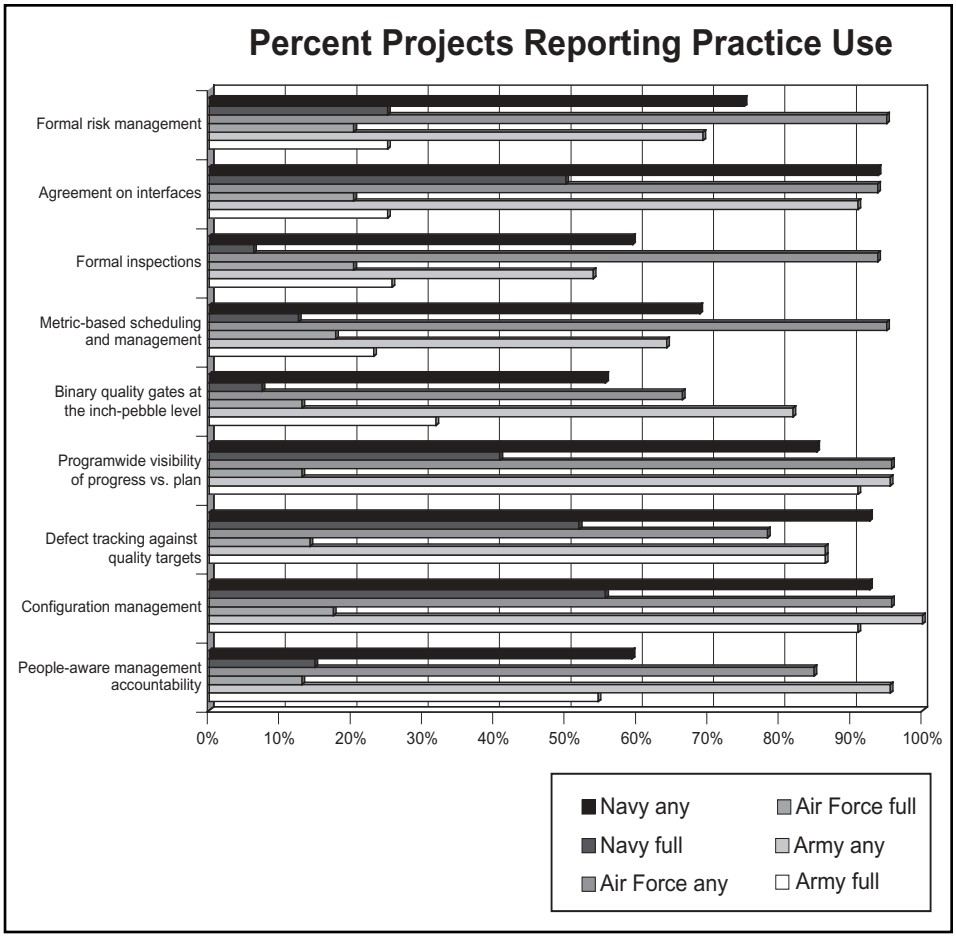


Figure 1: *Comparison of Service Adoption Rates*

References

1. Department of Defense. Defense Software: Review of Defense Report on Software Development Best Practices. Washington, D.C.: General Accounting Office, 2000.
2. Department of Defense. Major Management Challenges and Program Risks: Department of Defense. Washington, D.C.: General Accounting Office, 2001.
3. Department of Defense. Best Practices: Successful Application to Weapon Acquisitions Requires Changes in DoD's Environment. Washington, D.C.: General Accounting Office, 1998.
4. Department of Defense, Office of the Inspector General. Summary of Audits of Acquisition of Information Technology. Washington, D.C.: General Accounting Office, 2000.
5. Charette, R., and J. J. McGarry. Tri-Service Assessment Initiative Phase 1 Systemic Analysis Executive Report. Washington, DC.: Tri-Service Assessment Initiative, 2001.
6. Department of Defense, Defense Science Board. Report of the Defense Science Board Task Force on Defense Software. Washington, D.C.: 2000.
7. Glennan Jr., T.K., et al. Barriers to Managing Risk in Large Scale Weapons System Development Programs. Santa Monica, Calif.: RAND, 1993: 30.
8. Eslinger, S. Software Acquisition and Software Engineering Best Practices. Los Angeles: The Aerospace Corporation, 1999: 40.
9. Anderson, M.H., and E. Rebentisch. "Commercial Practices – Dilemma or Opportunity?" Program Manager 27.2 (1998): 16-21.
10. Software Program Managers Network. "Program Managers Guide to Software Acquisition Best Practices." Ver. 5.

Integrated Computer Engineering 1999.
11. Software Program Managers Network. "16 Critical Software Practices For Performance-based Management." Ver. 5.1. Integrated Computer Engineering 1999.
12. Hilburn, T., et al. FAA-ARA Software Engineering Competency Study: Final Report. Daytona Beach, Fla.: Embry-Riddle Aeronautical University, 1998.

Notes

1. The eight practices studied were past performance, best value, commercial warranties, government/contractor relationship, performance-based specifications, commercial specifications and standards, streamlined contract administration, and use of commercial off-the-shelf/non-developmental item components. The study also found considerable benefits with few drawbacks for using the practices.
2. The nine Airlie Practices were established in 1995 by a group of experts convened by the Navy's Software Program Manager's Network at the Airlie Center outside Warrenton, Va. (now historically referred to as "The Airlie Council").
3. ACAT I is defined as an acquisition program that is not a highly sensitive classified program, and that is designated as a Major Defense Acquisition Program; or estimated to require an eventual total expenditure for research, development, test, and evaluation of more than $355 million in fiscal year (FY) 1996 constant dollars; or for procurement of more than $2.135 billion in FY 1996 constant dollars. ACAT II is defined as those acquisition programs that do not meet the criteria for an ACAT I program, but are estimated to require an eventual total expenditure for research, development, test, and evaluation of more than $135 million in FY 1996 constant dollars; or for procurement of more than $640 million in FY 1996 constant dollars; or if designated as major by the DoD component head. ACAT III is defined as those acquisition programs that do not meet the criteria for an ACAT I or ACAT II. Other is defined as any acquisitions not designated with an ACAT level.

About the Author

Richard Turner, D. Sc., is a member of the Engineering Management and Systems Engineering Faculty at The George Washington University in Washington, DC. Currently, Dr. Turner is the assistant deputy director for Software Engineering and Acquisition in the Software Intensive Systems Office of the Under Secretary of Defense (Acquisition, Technology & Logistics). Dr. Turner is co-author of the book *CMMI Distilled* published by Addison Wesley.

1931 Jefferson Davis Highway
Ste. 104
Arlington, VA 22202
Phone: (703) 602-0851 ext. 124
E-mail: rich.turner@osd.mil

Achieving CMMI Level 5 Improvements with MBASE and the CeBASE Method

Dr. Barry Boehm, Dr. Daniel Port, and Apurva Jain
University of Southern California

Dr. Victor Basili
University of Maryland

Tuesday, 30 April 2002
Track 1: 1:00 - 1:40
Ballroom A

Each branch of service in the Department of Defense has major initiatives to pursue more advanced software-intensive systems concepts involving network-centric warfare with self-adaptive networks and cooperating human and autonomous agents. The ability to balance discipline and flexibility is critically important to developing such highly dependable software-intensive systems in an environment of rapid change. Risk-management orientation enables users of Capability Maturity Model® Integration℠ (CMMI℠) to apply risk considerations to determine how much discipline and how much flexibility is enough in a given situation. The risk-driven nature of the spiral model and MBASE enables them to achieve a similar balance of discipline and flexibility. When these project-level approaches are combined with the organization-level approaches in the Experience Factory, the result is the unified Center for Empirically Based Software Engineering (CeBASE) method described in this article.

Recent events in Afghanistan have convincingly demonstrated the value of software and information technology in achieving military superiority. Each of the Department of Defense (DoD) services has major initiatives to pursue even more advanced software-intensive systems concepts involving network-centric warfare with self-adaptive networks and cooperating human and autonomous agents.

However, there are tremendous challenges in providing the software product and process capabilities necessary to realize these concepts. These challenges include security, scalability, interoperability, legacy systems transition, uncontrollable commercial off-the-shelf (COTS) products, and synchronizing dozens if not hundreds of independently evolving systems in a world of increasingly rapid change.

In particular, the processes for managing these complex systems of systems will require both highly disciplined methods to ensure dependable operations and highly flexible methods to adapt to change.

Fortunately, DoD's new 5000-series policies on evolutionary acquisition and spiral development provide the acquisition and program management framework to achieve this balance of discipline and flexibility. Also, the recent Capability Maturity Model® (CMM®) Integration℠ (CMMI℠) provides a development framework for integrating software and systems consideration with degrees of freedom for tailoring development processes to achieve appropriate balances of discipline and flexibility. However, these initiatives fall short of providing specific techniques for achieving and maintaining the right balance of discipline and flexibility for a particular program's evolving situation.

In our previous three CrossTalk articles, we provided some specific methods that programs can use to achieve this balance. In "Understanding the Spiral Model as a Tool For Evolutionary Acquisition [1]," we showed how appropriate use of the spiral model enables programs to achieve the flexibility needed for evolutionary acquisition, while applying risk-management principles to retain an appropriate level of program discipline.

"... the opportunity is here for other organizations to use the Experience Factory approach to achieve CMMI Level 5 benefits well before reaching Level 4."

In "Balancing Discipline and Flexibility with the Spiral Model and MBASE [2]," we showed how risk considerations could be used to realize appropriate but different process models for different program situations. We also elaborated on some of the specific practices in Model-Based (system) Architecting and Software Engineering (MBASE) such as the use of life-cycle anchor-point milestones to keep the program on track during its evolution.

In "Using the Spiral Model and MBASE to Generate New Acquisition Process Models: SAIV, CAIV and SCQAIV [3]," we showed how programs could use MBASE risk management techniques to avoid many overruns of fixed schedules and budgets. This is done by prioritizing desired features and inverting the development process to deliver the most important features within the available schedule or budget.

MBASE and the CeBASE Method

However, the spiral, MBASE, and Schedule as Independent Variable (SAIV) approaches all operate at the individual project level. This still leaves open the coverage of the counterpart CMMI organization-level process areas, particularly those of achieving continuous improvement of the organization's processes.

In this article, we show how MBASE has been integrated with the University of Maryland's (UMD) organization-level Quality Improvement Paradigm (QIP), Experience Factory (EF), and Goal-Question-Metric (GQM) approaches into a Center for Empirically Based Software Engineering (CeBASE) method, which successfully addresses these challenges. CeBASE is sponsored by the National Science Foundation, NASA, and the DoD, and jointly led by the UMD and the University of Southern California (USC).

As we explored the details of Maryland's QIP, EF, and GQM approaches and USC's MBASE approach, we found that they were expressing very similar principles and practices. The Spiral Model's initial focus on system objectives was consistent with the QIP's initial focus on organizational and project-specific goals expressed in

® Capability Maturity Model, CMM, Software Capability Maturity Model, and SW-CMM are registered in the U.S. Patent and Trademark Office.
℠ CMM Integration and CMMI are service marks of Carnegie Mellon University.

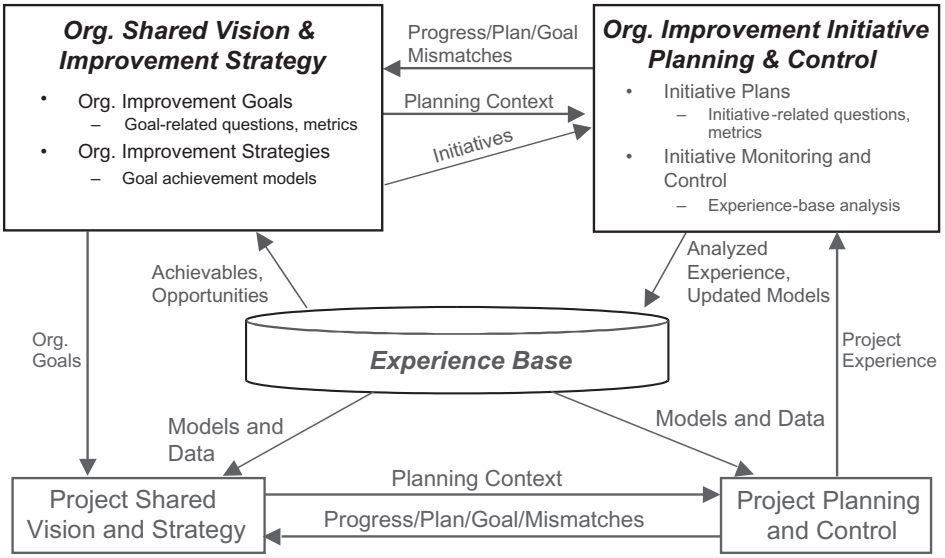


Figure 1: *Experience Factory Framework*

context using the GQM approach. The EF's focus on organizational learning to understand a system's operational stakeholders and their goals corresponds strongly with MBASE's stakeholder win-win approach to mutual stakeholder understanding and development of a shared system vision.

In the next section of this article, we summarize the key principles and practices of the QIP, the EF, and the GQM approaches; provide evidence of their successful application over 25 years of practice in the NASA Goddard-University of Maryland-Computer Science Corp. (CSC) Software Engineering Laboratory (SEL); and provide an example of their application at a systems as well as software level. In the section "The CeBASE method and CMMI," we present the CeBASE method and show how its process elements cover the process areas of the CMMI. In "Using the CeBASE Method," we show a version of the CeBASE method that has been successfully applied to more than 100 electronic services applications over six years' practice at USC.

Our conclusions include a diagram summarizing the process model distinctions among traditional approaches such as the Waterfall Model and Software Capability Maturity Model® (SW-CMM®); project-oriented approaches such as the spiral model, MBASE, and the Rational Unified Process (RUP); and integrated project/organization approaches such as the CMMI and CeBASE Method.

The QIP, GQM, and EF Approach

Framework and Methods

Since 1976, the UMD has been collaborating with NASA-Goddard and CSC on the SEL. The UMD and the SEL have developed and refined a series of closed-loop feedback processes that have resulted in significant improvements in software quality across more than 100 large software applications in the last 25 years.

The formulation of these feedback processes is called the QIP [4]. It uses six steps to provide an organized approach to continuous software quality improvement: 1) characterizing the organization, 2) setting goals, 3) choosing and instrumenting an appropriate process, 4) executing and monitoring the process, 5) analyzing the data to identify improvements, and 6) packaging the experience and improvements for future use.

The QIP makes use of the GQM approach, which is a mechanism for defining and evaluating a set of operational goals using measurement [5]. It ensures that your general goals are elaborated into specific questions and metrics for tracking progress and evaluating success, and that your people do not waste effort collecting and analyzing data weakly related to your goals.

The GQM approach can be applied at both the project level and the organization level. The EF [6] provides a consistent way of operating at both levels, as shown in Figure 1. Organization and project goals are determined by involving the relevant success-critical stakeholders in negotiating mutually satisfactory (win-win) and achievable goals.

For example, the organization may set a goal to reduce its projects' software cycle time by 50 percent. The initial implementing project may set goals and plans to have each project activity reduce its calendar time by 50 percent. As the project proceeds, its progress is monitored for progress/plan/goal mismatches, as shown at the bottom of Figure 1. While design, code, and test planning may finish in 50 percent less time, integration and test may start showing a 50 percent increase rather than decrease in duration. Analyzing this progress/plan/goal mismatch would determine the root cause to be delays due to inadequate test planning and preparation of test tools, test drivers, and test data. Further, shortening the test plan activity had produced no cycle timesaving, as test planning was not on the project's critical path.

The results of this analysis would be fed into the organization's experience base: Future cycle-time reduction strategies should focus on reducing the duration of critical path activities, and options for doing this include increasing the thoroughness and duration of noncritical-path activities. Overall then, as shown in the center of Figure 1, the EF analyzes and synthesizes such kinds of experience, acts as a repository for the experiences, and supplies relevant experience to the organization on demand. The EF packages experience by building informal and formal models and measures of various processes, products, and other forms of knowledge via people, documents, and automated support.

QIP, GQM, and EF in Practice

The application of the integrated set of these methods is referred to as the Experience Factory Organization, which resulted in a continuous improvement in software quality and cost reduction during the quarter-century life span of the SEL. When measured during three baseline periods in 1987, 1991, and 1995 (each representing about three years of development efforts), the demonstrated improvements included a 75 percent decrease in development defect rates from 1987 to 1991, and a 37 percent decrease from 1991 to 1995. We also observed a reduced development cost of 55 percent from 1987 to 1991 and of 42 percent from 1991 to 1995 [7, 8].

A more detailed example of improvement over time, in Figure 2, shows the defect rates in defects per thousand delivered lines of code (K-DLOC) for similar classes of projects at CSC during the application of the EF concepts. Over time, the defect models became well established and the range of variation (indicated by the upper and lower lines) narrowed, allowing managers to better manage quality [9]. Thus, the EF approach enabled the SEL portion of CSC to achieve SW-CMM Level 5 improvements well before CSC became a Level 5 organization in 1998. With the CMMI's emphasis on measurement and analysis as early as Level 2, the opportunity is here for other organizations to use the EF approach to achieve CMMI Level 5 benefits well before reaching Level 4.

Various EF concepts have been successfully applied in other organizations, including Daimler Chrysler, Robert Bosch, TRW, and Allianz.

Applying EF Concepts at the Systems Level

Systems-Level Goals and Questions

Many software organizations interpret the EF concepts at just the software level. They miss many opportunities to reap much more significant returns on investment at the systems level. For example, suppose that your software intensive project has a proposed goal for an improvement initiative to reduce the project's software defect rates. What should be your next step? Usually it would be to look down from the software goal into the software details. How will we define defect? What are the counting rules for overlapping defects? What are our current defect rates?

With EF at the systems level, your next step is to look upward and sideways from the software and ask system-level questions: Why do we want to reduce software defect rates? What system goals are being frustrated by software defects? Where are the frustrations the greatest?

For example, in an operational order-processing system, the answers may be that the software defects are causing 1) too much downtime in the operation's critical path, 2) too many defects in the system's operational plans, and 3) too many new-release operational problems.

These insights enable you to reformulate your improvement initiative goal to decrease the organization's software defect-related losses in operational cost effectiveness. Items one through three become initial high-payoff target sub-goals for the initiative. Given this new goal and context, what should be your next step?

Sub-Goal Level Questions, Models, and Metrics

Again, a good next step is to ask *why* the software defects are causing operational problems, often with the help of models. For example, Figure 3 shows a critical-path model for analyzing the order-processing downtime and delays caused by software defects. Analyzing this model may lead to several valuable insights, improvement strategies, and system payoffs:

1. Often, major sources of delay are additional manual processing delays caused by software or non-software problems, as with the Scientific American order processing system discussed in Boehm [10].
2. The logic for packaging and delivery scheduling can become quite complex when only part of an order is in stock. (It is generally okay to send a partial shipment at Amazon.com, however, not for jet engine repair spare parts.) Software defects can again cause considerable operational delays.
3. "Produce status reports" defects should not be on the operational critical path. This module was probably put on the critical path by a programmer's detailed design coupling and cohesion decision without considering its potential system effect, resulting in status-report defects causing order-delivery delays.
4. The overall legacy order-processing system may just be too slow and difficult to modify and should be replaced downstream by a new Web-based order-processing system. It is generally good to be asking *why not* as well and *why* questions.

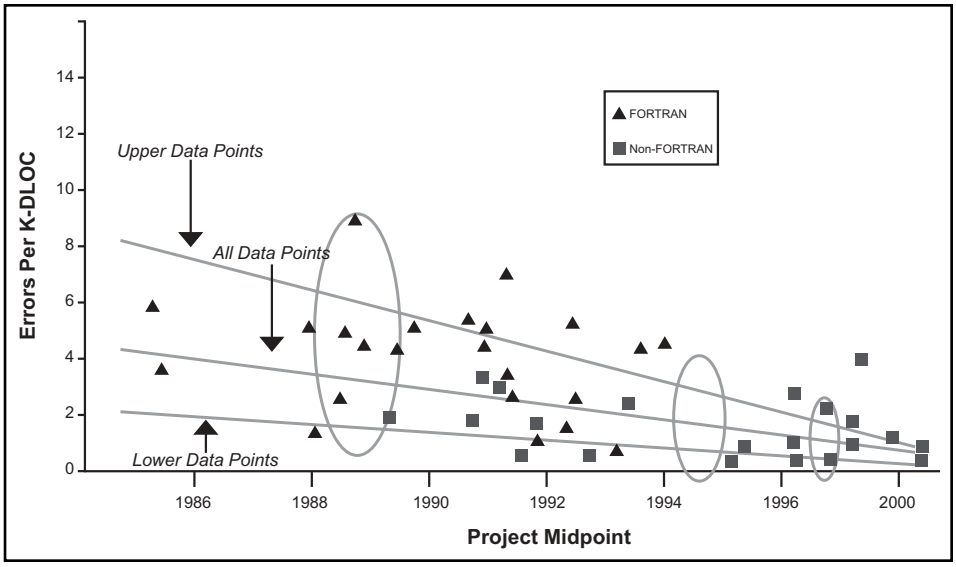


Figure 2: *Defect Rate Improvements in Software Engineering Laboratory/Computer Science Corporation Projects*

Putting It All Together

Each of these sub-goal-related initiatives needs to be monitored and controlled with respect to improvement-related metrics such as order-processing cycle time and user satisfaction. The results need to be integrated with other ongoing improvement initiatives to ensure synergy and integration with the overall organizational experience base discussed earlier in the "framework and methods" section. The sidebar on page 12 shows the resulting systems-level EF-GQM initiative steps.

These system-level EF-GQM approaches are already being practiced by leading-edge software organizations. Two of the recent Institute of Electrical and Electronics Engineers' Software Process Achievement Award winners, Advanced Information Services, Inc. (AIS) and Tinker Air Force Base, are good examples [11, 12].

AIS uses Balanced Scorecard techniques to integrate its software, systems, project, and organizational goals in such areas as customer satisfaction, financial performance, employee growth, process improvement, and organizational learning capability. Specifically, AIS periodically assesses its performance and rate of progress in these areas on a Balanced Scorecard form and uses the results to adjust its improvement efforts in each area. Tinker has used its software insights to stimulate systems-level initiatives with its counterpart hardware and test organizations to improve system-level cycle time and to deliver quality in such areas as B-2 Test Program Sets.

This kind of approach is what transitioning from the software CMM to the CMMI is all about. It requires software organizations to be more pro-active than reactive in interacting with the operational system stakeholders. It gets software people applying their necessary expertise to system issues. It results in much larger bottom-line payoffs for the operational system stakeholders. The next two sections discuss how the CeBASE method integrates software and system-level activities as well as project- and organizational-level activities and how its practices map to the process areas and practices in the CMMI.

Figure 3: *Order-Processing System Critical Path Model*

Systems-Level Experience Factory Goal-Question-Metric Initiative Steps

1. Identify a software-related improvement initiative goal.
2. Relate this to system-level goals: Ask questions about why the initiative is needed.
3. Use the results to identify the related system-level improvement initiative goal and high-payoff sub-goal initiatives.
4. Perform a systems-level root-cause analysis: Construct relevant models, ask questions about why the current-system shortfalls cause problems and whether or not to try alternative system approaches.
5. Identify the system improvement initiative's key stakeholders; achieve a shared vision of and commitment to the initiative goals and strategies.
6. Establish improvement initiative plans and progress metrics for each sub-goal initiative and the overall initiative.
7. Execute, monitor, and control the initiative plans with key stakeholder participation. Feed the resulting experiences into the organization's experience base for future benefits.

The CeBASE Method and the CMMI

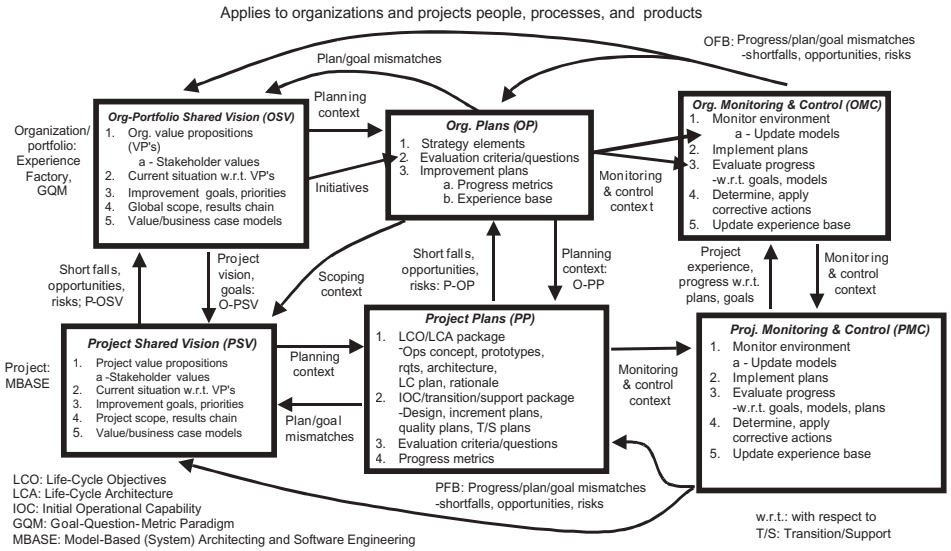
The CeBASE Method Framework

Overall, we found that both EF-GQM and MBASE could be integrated into a common CeBASE method. Its framework is organized around a trio of common strategic themes, shown by the vertical pairs in Figure 4. These three themes are the stakeholders' *shared vision* for the organization or project; risk-driven *plans* for process, product, and people; and continuous *monitoring and control*. As seen in Figure 4, these themes express both the operation of EF-GQM at the organizational level and the operation of MBASE-GQM at the project level. Within a large diverse organization, we may wish to consider a particular set of projects within a *portfolio* or product line of related products or services.

To start at the upper left of Figure 4, the organization's value propositions are often contained in an organizational mission statement. This will cover the organizational stakeholders' agreed-upon win conditions and will be expressed in terms of such Balanced Scorecard elements as customer satisfaction, financial performance, employee growth, process improvement, and organizational learning capability.

Improvement goals and priorities will come from Balanced Scorecard assessments. These might include such goals as reducing software development cycle time or reducing average order-delivery time. The specific quantitative goals, e.g., reduce software development cycle time by 50 percent, would be based on initial cost/value analyses. These are developed using such techniques as the DMR Consulting Group's Results Chains linking improvement initiatives to contributions and benefits-realized outcomes [13] and associated business-case models linking the value of benefits realized to the costs invested in the initiatives.

Figure 4: *The CeBASE Method Framework*



The CeBASE Method at the Organizational Level

The resulting organization (or portfolio) shared vision (OSV) (Figure 4) drives two sets of initiatives. Horizontally in Figure 4, it drives initiatives to improve software cycle time or to reduce order-delivery time across the organization. These initiatives will have strategy elements and their associated organization-level improvement plans (OP) such as reducing delays in order-delivery time due to software defects.

Following the GQM paradigm, the *goal* of reducing order-delivery time is related to organization plan *questions* such as, "What is our current record on delivery times?" "What distinguishes orders with significantly better or worse delivery times?" "What are the costs and benefits resulting from an improvement initiative?"

These questions are related to *metrics* such as overall order-delivery time, critical-path task times, costs and benefits of eliminating related software defects, and customer order-delivery satisfaction ratings. These are used to monitor and control organization-level progress (OMC), and to adjust the strategies and goals based on the organizational feedback of progress/plan/goal achievements and mismatches.

The CeBASE Method at the Project Level

Vertically in Figure 4, the OSV drives the nature of each project's shared vision (PSV) and its associated goals and priorities. Thus, for example, an organizational improvement goal to reduce software development cycle time by 50 percent will be reflected in the project's value propositions and improvement goals or the organization project shared vision (O-PSV) (see arrow in Figure 4). This will lead to project-level goals, models, questions, and metrics such as reducing the duration of each project task by 50 percent.

This in turn leads horizontally across the bottom of Figure 4 to project-level plans (PP), project monitoring and control activities (PMC), and to the determination and feedback of project-level progress/plan/goal mismatches. This mismatch feedback could be negative such as increased integration and test task durations or it could be positive. For example, the project might incorporate a new in-transit-visibility COTS package for order delivery tracking that both helps in delay diagnosis and improves customer satisfaction by answering questions about delivery delays.

This project feedback propagates upward to the organizational level along all three lines of traceability. The shortfalls or

opportunities with respect to organizational shared vision and goals are fed back along the project OSV (P-OSV) arrow at the left. The corresponding plan-context feedback occurs along the project OP (P-OP) arrow in the center, and the monitoring and control feedback at the right is used to update the organization's detailed experience base on best practices for achieving goals.

As a final observation, note that the content of the PP element consists of the Spiral/MBASE Life-Cycle Objectives (LCO), Life-Cycle Architecture (LCA), and Initial Operational Capability (IOC) anchorpoint milestone content that we discussed in our May 2001 and December 2001 CrossTalk articles [1, 2]. Thus, the Spiral/MBASE guidelines in those articles have become the project-level guidelines for the CeBASE method. A detailed example of these guidelines is shown next.

CeBASE Guidelines Example: Shared Vision

The CeBASE project-level and organization-level shared vision guidelines are quite similar. Their main difference is one of context: The project-level shared vision has the organization-level shared vision as context and shows traceability to it, but not vice versa. Figure 5 shows the table of contents and example text from the project-level shared vision guidelines. In the CeBASE method, it is the first item to be drafted by the project's Integrated Product Team of success-critical stakeholders or its equivalent. It sets the stage for subsequent Inception Phase prototyping and stakeholder win-win requirements negotiation.

The shared vision guidelines are adopted from best commercial practices in ways that apply to public service applications as well. The system capability "elevator" description comes from Geoffrey Moore's classic *Crossing the Chasm* [13]. The "Benefits Realized" and "Results Chain" sections are adapted from the DMR Consulting Group's Benefits Realization Approach [14]. The Results Chain identifies the full set of initiatives necessary to realize the proposed system's benefits; this also identifies the full set of success-critical stakeholders who should be involved in the system's definition. The current version of the guidelines is at <http://cebase.org/cebase method>.

Table of Contents

2.1 System Capability Description

A concise description of the system that can pass the "elevator test" described in Geoffrey Moore's *Crossing the Chasm* [13]. This would enable you to explain why the system should be built to an executive while riding up or down an elevator. It should take the following form:

- For (target customer)
- Who (statement of the need or opportunity)
- The (product name) is a (product category)
- That (statement of key benefit-that is, compelling reason to buy)
- Unlike (primary competitive alternative)
- Our product (statement of primary differentiation)

Here is an example for a corporate order-entry system: "Our sales people need a faster, more integrated order-entry system to increase sales and customer satisfaction. Our proposed Web order system would give us an e-commerce order-entry system similar to Amazon.com's that will fit the special needs of ordering mobile homes and their aftermarket components. Unlike the template-based system our main competitor bought, ours would be faster, more user friendly, and better integrated with our order fulfillment system."

Common Pitfalls:

- Not relating the need or opportunity to the goals in the organization's Shared Vision.
- Being too verbose about "our product" or its key benefits.

2.2 Key Stakeholders

Identify each stakeholder by their home organization, their authorized representative for project activities, and their relation to the Results Chain. The four classic stakeholders are the software/IT system's users, customers, developers and maintainers. Additional stakeholders may be system interfacers, subcontractors, suppliers, venture capitalists, independent testers, and the general public (where safety or information protection issues may be involved).

Common Pitfalls:

- Being too pushy or not pushy enough in getting your immediate clients to involve the other success-critical stakeholders. Often, this involves fairly delicate negotiations among operational organizations. If things are going slowly and you are on a tight schedule, seek the help of your higher-level managers.
- Accepting just anybody as an authorized stakeholder representative. You don't want the stakeholder organization to give you somebody they feel they can live without. Some good criteria for effective stakeholder representatives are that they be empowered, representative, knowledgeable, collaborative, and committed.

Figure 5: *Example CeBASE System-Level Shared Vision Content*

CeBASE Method Coverage of the CMMI

Example Mapping: Requirements Development

To test its coverage of critical issues, we have done a mapping of the CeBASE method onto the CMMI's 24 process areas using the CMMI summary tables in Ahern, et al. [15]. A mapping example is provided in a longer version of this paper available at <http://www.cebase.org>.

Overall, the mapping indicated that the CeBASE method covered the CMMI goals and practices well. It provided the CeBASE team with some action items to address missing elements covered in the CMMI. Most significantly, though, it identified items that we have found important to software and systems engineering that were missing in the CMMI. These included a business case justifying the need for required features, having a stakeholder win-win prioritization of requirements (for coping with new requirements and fixed budgets or schedules), coverage of project requirements (required platforms, resource constraints), level of service requirements (the -ilities), and evolution requirements (to avoid point-solution architectures).

Overall CeBASE Method Coverage of the CMMI

Overall, we found not only a strong correspondence but also an almost complete coverage of the CMMI's practices by the organizational and project components of the CeBASE method. We are extending the CeBASE method to cover the specific CMMI processes not currently covered. A summary of the percentage of the CMMI process areas covered by the CeBASE method is shown in Table 1 (see page 14). The "+" annotations in Table 1 indicate that the CeBASE method's coverage goes considerably beyond that of the CMMI. For example, it covers not just an organizational process focus but also an organizational product and people focus. The "-" annotations in Table 1 indicate that some areas in the CeBASE method still remain to be fleshed out, such as detailed guidelines for organizational training plans, although they are covered in principle.

The CeBASE method also provides a prescriptive approach for an organization to use in tailoring the CMMI's generic practices to its particular culture, environment, and value propositions. Thus, an e-commerce organization's value propositions (rapid time to market, rapid adaptation to change) will

Process Management • Organizational Process Focus: 100+ • Organizational Process Definition: 100+ • Organizational Training: 100- • Organizational Process Performance: 100- • Organizational Innovation and Deployment: 100+	**Engineering** • Requirements Management: 100 • Requirements Development: 100+ • Technical Solution: 60+ • Product Integration: 70+ • Verification: 70- • Validation: 80+
Project Management • Project Planning: 100 • Project Monitoring and Control: 100+ • Supplier Agreement Management: 50- • Integrated Project Management: 100- • Risk Management: 100 • Integrated Teaming: 100 • Quantitative Project Management: 70-	**Support** • Configuration Management: 70- • Process/Product Quality Assurance: 70- • Measurement and Analysis: 100- • Decision Analysis and Resolution: 100- • Organizational Environment for Integration: 80- • Causal Analysis and Resolution: 100

Note: All amounts are percentages.

Table 1: *CeBASE Method Coverage of CMMI*

cause it to adopt more flexible processes. However, such elements as the anchor-point milestones will balance this flexibility with sufficient discipline to keep the overall process under control. The value propositions of an organization developing safety-critical products or services will cause it to emphasize more rigorous specifications, processes, and practices, but in ways that enable it to cope with rapid change. Examples include capturing evolution requirements, designing systems to accommodate future change, building in buffer periods to synchronize and stabilize processes [16], or to adapt to potential schedule or budget slips by dropping lower-priority product features [3].

Another point worth emphasizing is that the EF component of the CeBASE method supports a continuous vs. staged approach to process improvement. You do not need to be a CMM Level 4 organization to begin realizing significant benefits from organizational innovation or causal analysis.

Using the CeBASE Method

Since 1996, we have been applying the EF and GQM approaches to improve the project-oriented MBASE aspects of the CeBASE method by using them to improve an annual series of USC electronic services projects. These are developed using annually improved MBASE guidelines by teams of five master's-level students as developers and staff members of USC's Information Services Division as clients (customers, users or user representatives, and maintainers). Each year, we have 15 to 20 teams execute the MBASE inception and elaboration phases in the fall semester to develop and validate life-cycle architecture packages for USC electronic services applications' candidates. The top six to 10 of these applications are then selected for spring semester teams who execute the MBASE construction and transition phases and deliver initial operational capability application systems.

Our shared vision for the USC Center for Software Engineering's research and education goals incorporates the win conditions of not only our students and their project clients, but also other stakeholders such as the center's staff and prospective technology users, represented by our 35 industry and government affiliate organizations [17]. Our questions and metrics include stakeholder critiques of each project and extensive instrumentation of the projects' effort, schedule, quality, productivity, and behavioral characteristics [18, 19, and 20].

Table 2 summarizes four years' experience to date in applying and refining CeBASE on an annual selection of real-client projects.

A few explanatory comments on Table 2 are in order. The number of LCA teams is larger than the number of IOC teams because USC's fall course is a core course for the USC master's of science degree in computer science and has a much larger enrollment than the spring course, which is only required for a few specialization areas. In 1996-97, the subset of projects to be continued in the spring was primarily those having students continuing from the fall course. After we found that most of the 1996-97 products went unused, we performed a critical success factor analysis and determined a set of spring project selection criteria (e.g., library commitment to product use, empowered clients) that increased the project adoption rate. Even then, unforeseen circumstances such as the inevitable changes in library infrastructure and organizational responsibilities have caused some applications' usage commitments to be overtaken by events. This is a frequent phenomenon for electronic services applications [21].

In general, the EF improvements on MBASE have effected a uniform improvement in outcome, but there are some anomalies. For example, the 12 percent of projects failing IOC in 1999-2000 were due to a team who botched their product transition when their client was unexpectedly called out of town during the transition period. Another example was our introduction of midcourse client briefings on core capability expectations in 1999-2000 as part of our SAIV process [3]. SAIV only guarantees the delivery of a highest-priority core capability set of features, with further features added as time is available. While this resulted in early client disappointments at LCA where client success scores dropped from 4.74 in 1998-1999 to 4.48 in 1999-2000, there was a dramatic increase in the clients' success score for the delivered product (4.3 in 1998-1999 to 4.75 in 1999-2000).

The 1998-1999 improvement in the "Failing LCO" criterion shown in Table 2 resulted primarily from our introduction of a simplifier and complicator (S&C) expectations management activity. This helped the developers to have a better understanding of the system and the stakeholders by leveraging an experience base of designs that help simplify the architecture (the *simplifiers*) and apply a risk-driven approach to the architectural areas that may cause significant complications (the *complicators*). Involving the clients in risk management activities throughout the process clearly contributed to their rating virtually all delivered applications as highly satisfactory.

Table 2: *Annual University of Southern California E-Services Project Outcomes*

	1996-1997	1997-1998	1998-1999	1999-2000
LCA Teams	15	16	19	22
Failing LCO	27%	25%	5%	5%
Failing LCA	0%	0%	0%	0%
LCA Client Score	4.46	4.67	4.74	4.48
IOC Teams	6	5	6	8
Failing IOC	16%	0%	0%	12%
IOC Client Score	n/a	4.15	4.3	4.75
IOC Regularly Used	16%	60%	50%	62%

Conclusions

Figure 6 summarizes the distinctions among maturity models such as the SW-

CMM and the CMMI; process models such as the waterfall model; and process model generators such as MBASE, RUP, and the CeBASE method. It shows where each model fits with respect to organizational focus (project vs. organization), application focus (software vs. system), and operational focus (practice vs. assessment).

From Figure 6, we can see that the SW-CMM covers both project and organization considerations, but it has shortfalls in both applications focus (software, not systems) and operational focus (assessment, not practice). We can also see that solutions focused on redressing one of the two shortfall dimensions will still have shortfalls of their own in another dimension. Thus, the CMMI redresses the systems shortfall in the SW-CMM, but it still has the shortfall of providing explicit guidelines for assessment but not for project practices. While MBASE and RUP provide explicit guidelines for project practices, they do not provide counterparts for an organization's practices. However, the combination of CMMI and the CeBASE method covers all aspects of operational, organizational, and application focus.

In terms of future software-intensive system challenges, the ability to balance discipline and flexibility is critically important to the development of highly dependable software-intensive systems in an environment of rapid change. The CMMI's risk-management orientation enables its users to apply risk considerations to determine how much discipline and how much flexibility is enough in a given situation. The risk-driven nature of the spiral model and MBASE enables them to achieve a similar balance of discipline and flexibility. When these project-level approaches are combined with the organization-level approaches in the EF, the result is the unified CeBASE method summarized in the section "The CeBASE Method and the CMMI." It currently implements most of the CMMI, is being extended to cover the full CMMI, and has a strong track record of continuous process improvement at USC's and UMD's Software Engineering Laboratories and industry adapters elsewhere.◆

Operational Focus: Assessment / Practice		**Organizational Focus** Project	Organization
Application Focus	Software	Software CMM / Waterfall, Incremental	Software CMM / Early EF, GQM
	Systems	CMMI / Spiral, MBASE, RUP	CMMI / CeBASE Method

Figure 6: *Process Model Coverage Distinctions*

Acknowledgements

We would like to acknowledge the support of the National Science Foundation in establishing CeBASE, the DoD Software Intensive Systems Directorate in supporting its application to DoD projects and organizations, and the affiliates of the USC Center for Software Engineering and the University of Maryland's software engineering program for their contributions to MBASE and CeBASE.

References

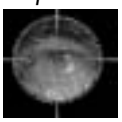
1. Boehm, B., and W. Hansen. "Understanding the Spiral Model as a Tool for Evolutionary Acquisition." CrossTalk May 2001.
2. Boehm, B., and D. Port. "Balancing Discipline and Flexibility with the Spiral Model and MBASE." CrossTalk Dec. 2001.
3. Boehm, B., D. Port, L. Huang, and A. W. Brown. "Using the Spiral Model and MBASE to Generate New Acquisition Process Models: SAIV, CAIV, and SCQAIV." CrossTalk Jan. 2002.
4. Basili, Victor R., and Gianluigi Caldiera. "Improve Software Quality by Reusing Knowledge and Experience." Sloan Management Review 37.1 (1995).
5. Basili, Victor R., Gianluigi Caldeira, and H. D. Rombach. "The Goal Question Metric Approach." Encyclopedia of Software Engineering. Ed. J. Marciniak. Wiley, 1994.
6. Basili, Victor R., Gianluigi Caldeira, and H. D. Rombach. "The Experience Factory." Encyclopedia of Software Engineering. Ed. J. Marciniak. Wiley, 1994.
7. Basili, Victor R., Gianluigi Caldiera, Frank McGarry, Rose Pajersky, Gerald Page, and Sharon Waligora. "The Software Engineering Laboratory – An Operational Software Experience Factory." 14th International Conference on Software Engineering. May 1992.
8. Basili, Victor R., Marvin Zelkowitz, Frank McGarry, Jerry Page, Sharon Waligora, and Rose Pajerski. "Special Report: SEL's Software Process-Improvement Program." IEEE Software 12.6 (1995): 83-87.
9. McGarry, F. "What Is A Level 5 Organization? Lessons from 10 Years of Process Improvement Experiences at CSC." Proceedings of the Twenty-Sixth NASA Software Engineering Workshop. Nov. 2001.
10. Boehm, B. Software Engineering Economics. Prentice Hall, 1981.
11. Ferguson, P., et al. "Software Process Improvement Works!" Advanced Information Services, Inc. CMU/SEI-99-TR-027, Nov. 1999.
12. Butler, K., and W. Lipke, "Software Process Achievement at Tinker Air Force Base, Oklahoma." CMU/SEI-2000-TR-014, Sept. 2000.
13. Moore, Geoffrey. Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers. New York: Harper Business, 1991: 161.
14. Thorp, J., and DMR Consulting Group. The Information Paradox, McGraw Hill, 1998.
15. Ahern, D., A. Clouse, and R. Turner. CMMI Distilled. Addison Wesley, 2001.
16. Cusumano, Michael A., and Richard W. Selby. Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People. New York: Simon & Schuster, 1996.
17. Boehm, B., A. Egyed, D. Port, A. Shah, J. Kwan, and R. Madachy. "A Stakeholder Win-Win Approach to Software Engineering Education." Annuals of Software Engineering 6 (1998): 295-321.
18. Egyed, A., and B. Boehm. "Comparing Software System Requirements Negotiation Patterns." Systems Engineering 2.1 (1999): 1-14.
19. Port, D., and B. Boehm. "Introducing Risk Management Techniques Within Project-Based Software Engineering Courses." Computer Science Education 2002 (to appear).
20. Majchrzak, A., and C. Beath. "A Framework for Studying Learning and Participation in Software Development Projects." Management Informational Systems Quarterly, under review.
21. Boehm, B., "Project Termination Doesn't Equal Project Failure." IEEE Computer Sept. 2000: 94-96.

COMING EVENTS

May 13-17
Software Testing Analysis and Review (STAREAST 2002)

Orlando, FL
www.sqe.com/stareast

June 3-6
Combat Identification Systems Conference



Colorado Springs, CO
www.usasymposium.com

July 18-20
Shareware Industry Conference
St. Louis, MO
www.sic.org

July 22-25
Joint Advanced Weapons Systems Sensors, Simulation, and Support Symposium (JAWS S3)
Colorado Springs, CO
www.jawswg.hill.af.mil

August 19-22
The Second Software Product Line Conference
San Diego, CA
www.sei.cmu.edu/SPLC2/

September 9-13
International Conference on Practical Software Quality Techniques and International Conference on Practical Software Testing Techniques 2002 North
St. Paul, MN
www.softdim.cim/psqt/

April 28-May 1, 2003
Software Technology Conference 2003

Salt Lake City, UT
www.stc-online.org

About the Authors

Barry Boehm, Ph.D., is the TRW professor of software engineering and director of the Center for Software Engineering at the University of Southern California. He was previously in technical and management positions at General Dynamics, Rand Corp., TRW, Defense Advanced Research Projects Agency, and the Office of the Secretary of Defense as the director of Defense Research and Engineering Software and Computer Technology Office. Dr. Boehm originated the spiral model, the Constructive Cost Model, and the stakeholder win-win approach to software management and requirements negotiation.

University of Southern California
Center for Software Engineering
Los Angeles, CA 90089-0781
Phone: (213) 740-8163
Fax: (213) 740-4927
E-mail: boehm@sunset.usc.edu

Daniel Port, Ph.D., is a research assistant professor of Computer Science and an associate of the Center for Software Engineering at the University of Southern California (USC). Dr. Port's previous positions were assistant professor of Computer Science at Columbia University, director of Technology at the USC Annenburg Center EC2 Technology Incubator, co-founder of Tech Tactics, Inc., and a project lead and technology trainer for NeXT Computers, Inc. He received a doctorate from the Massachusetts Institute of Technology in applied mathematics with an emphasis on theoretical computer science in 1994 and a bachelor's degree in mathematics from the University of California in Los Angeles.

University of Southern California
Center for Software Engineering
Los Angeles, CA 90089-0781
Phone: (213) 740-7275
Fax: (213)740-4927
E-mail: dport@sunset.usc.edu

Apurva Jain is a doctoral student at the University of Southern California's Center for Software Engineering. Previously he was a project manager at SpruceSoft Inc. His research interests are software process management and pervasive computing. He received a bachelor's degree from Curtin University of Technology, Perth, Australia and a professional honors diploma from Informatics, Singapore.

University of Southern California
Center for Software Engineering
941 W. 37th Place, SAL 329
Los Angeles, CA 90089-0781
Phone: (213) 740-6505
Fax: (213) 740-4927
E-mail: apurvaj@sunset.usc.edu

Victor R. Basili, Ph.D., is a professor of Computer Science at the University of Maryland, the Executive Director of the Fraunhofer Center, Maryland, and one of the founders and principals in the Software Engineering Laboratory. He works on measuring, evaluating, and improving the software development process and product and has consulted for many organizations. Dr. Basili is a recipient of a 1989 NASA Group Achievement Award, a 1990 NASA/GSFC Productivity Improvement and Quality Enhancement Award, the 1997 Award for Outstanding Achievement in Mathematics and Computer Science by the Washington Academy of Sciences, and the 2000 Outstanding Research Award from ACM Special Interest Group on Software Engineering.

Computer Science Department
4111 AV Williams Building
University of Maryland
College Park, MD 20742
Phone: (301) 405-2668
Fax: (301) 405-2691
E-mail: basili@cs.umd.edu

U.S. Defense Department Requirements for Information Security

Kevin J. Fitzgerald
Oracle Corporation

Wednesday, 1 May 2002
Industry Plenary: 8:00 - 8:45
Ballroom

The events of 9/11 have resulted in a heightened awareness of the importance of information security. Paradoxically, there is a greater requirement than ever before for military, intelligence, and law enforcement entities to share increasingly sensitive data, yet enforce stringent information security policies to protect their critical infrastructure. The security of these information systems must meet the technical challenges of a diverse user community: need-to-know enforcement, interoperability, secure communications, and high availability, as well as offering independent measures of assurance required by federal directives (e.g., National Security Telecommunications and Information Systems Security Policy No. 11).

The importance of information security has increased substantially in the past few years, primarily due to the growth of the Internet. The events of 9/11 have resulted in a heightened awareness of the importance of information security in several key ways.

There is a greater need for information sharing than ever before to enable disparate intelligence, military, and law enforcement groups to selectively share information, yet maintain "need-to-know" provisions required by national security. There is an increased awareness of the threats posed by information warfare; without ever firing a shot, enemy forces could launch a cyberattack on a nation's critical infrastructure, thus rendering a foe helpless.

The importance of information security to the U.S. armed forces is thus both old and new. Its importance is old in that the problems of information security are, as they have always been, related to the confidentiality, integrity, and availability of information (the "CIA" of traditional information security). Its importance is new in response to cyberwarfare threats, the sheer volume of computerized information, and the numbers of people accessing it. U.S. Department of Defense (DoD) requirements for information security include all of the following:

- Large diverse worldwide user community.
- Coalition forces' need for interoperability.
- Enforce "need-to-know" while enabling greater data sharing.
- Highly secure communications.
- Stringent auditing requirements.
- Users access to multiple systems to carry out their mission.
- Critical nature of many defense systems requires 100 percent uptime.
- Independent measures of assurance as required by federal directives.

An explanation of each information security requirement follows below.

DoD Information Security Requirements

Large User Communities

The DoD represents an extremely large diverse worldwide user community. The sheer size of the user community accessing defense systems via the Web not only increases the risk to those systems, but also constrains the solutions that can be deployed to address that risk. Moving applications to the Web creates challenges in terms of scalability of security mechanisms, management of those mechanisms, and the need to make them standard and interoperable. Whereas the largest traditional enterprise systems typically supported hundreds of users, many Web-enabled defense systems have potentially thousands of users.

"Without ever firing a shot, enemy forces could launch a cyberattack on a nation's critical infrastructure, thus rendering a foe helpless."

Interoperability

Unlike traditional defense systems where a command or program owns and controls all components of the system, Web-enabled systems must exchange data with systems owned and controlled by others, e.g., other commands, suppliers, coalition forces, partners, etc. Security mechanisms deployed in these systems must therefore be standards based, flexible, and interoperable to ensure that they work with others' systems. They must support thin clients and work in multitier architectures.

Enforce Need to Know

Allowing greater access to data while enforcing need to know means that access control must be enforced on the data to ensure that the same security policy is enforced regardless of the method of data access. This requires a high degree of granularity in traditional access control mechanisms, as well as the ability to compartmentalize data access based on an application-specific security classification. (*Application-specific* meaning that organizations may have different data labeling requirements, and thus cannot necessarily use a fixed-labeling scheme across every organization that needs access to the same data). Real-time information sharing requires real-time or near real-time reclassification of data, so that, as threats change, information can be both shared and segmented among the multiple constituencies who need access to it.

Secure Communications

The sensitive nature of communications within the armed forces requires that even ordinary communications provide encryption for confidentiality and data integrity to ensure that communications are neither intercepted nor modified in transmission. Furthermore, large-scale encryption of stored data is generally a bad idea (as it does not address access control issues, gives users a false sense of security,

and slows down system performance). However, there is a requirement for selective encryption of some stored data as an extra layer of protection, for *defense-in-depth*.

Stringent Auditing

The more sensitive the data, and the more users with access to that data, the greater the requirement to hold users accountable through auditing. Unfortunately, a number of serious security breaches involving national security might have been prevented had proper auditing mechanisms been enforced. Auditing must be granular enough to focus upon a particular activity, user, or object, and comprehensive enough to record *all* user activity of interest, yet have minimal impact on performance. Auditing must also be tied into an alert mechanism to provide administrators with timely information.

User Access to Multiple Systems

Traditional mechanisms used to identify users and manage their access like granting each user an account and password on each system they access are not practical in a large interconnected environment such as organizational intranets or the Internet. It rapidly becomes too difficult and expensive for system administrators to manage separate accounts for each user on every system. There is a greater requirement for both strong user authentication – due to the increased amount of information users are able to access – and central identification and management of users due to the prohibitive cost of managing access for thousands if not hundreds of thousands of users across multiple systems. Furthermore, in the case of non-centralized account and privilege administration, shutting down or restricting a user's access in the event of a suspicious security event or security breach is time consuming. It also exposes the systems to additional breaches while the administrator is required to access and modify each separate system.

Availability

System availability is critical due to the nature of the mission of the U.S. armed forces. More than perhaps any other consumers of information technology, DoD systems require 100 percent uptime. For most commercial organizations, information unavailability during system downtime may be inconvenient and costly but not life threatening. For the armed forces, information availability may literally be the difference between mission success or failure and life or death.

Information Assurance

U.S. federal directives such as the National Security Telecommunications and Information Systems Security Policy (NSTISSP) No. 11, (see <www.nstissc.gov/Assets/pdf/nstissp11.pdf>) require information systems that access or manage information related to national security to have *independent* measures of information assurance, as evidenced by formal, independent (third-party) security evaluations. Acceptable criteria against which products may be evaluated include the Common Criteria ISO-15408 (see <http://csrc.nist.gov/cc/ccv20/ccv2list.htm>), the de facto worldwide evaluation criteria, and the Federal Information Processing Standard (FIPS)-140 (see <http://csrc.nist.gov/cryptval>), which attest to the correctness of cryptographic mechanisms.

In the past, procurement vehicles specified formal security evaluations. Many requiring a solution compliant with the Trusted Computer Security Evaluation Criteria (see <www.radium.ncsc.mil/tpep/library/tcsec>) or an Evaluation Assurance Level 4 (EAL4), as defined in the Common Criteria, were often granted waivers for this requirement based on functionality requirements that were not supported in the evaluated versions. Procurement waivers (from NSTISSP No. 11 requirements) will likely – and rightly – be much harder to acquire in a security environment after 9/11. Security will play a stronger role in the tradeoff analysis between security and functionality.

"The more sensitive the data, and the more users with access to that data, the greater the requirement to hold users accountable through auditing."

This article describes both the appropriate technical measures as well as specific security mechanisms that can address the above requirements (in general terms). Since many Web-based information-processing systems are built on database management systems, the technical solutions will be presented in terms of the protection of information stored in database systems.

Technical Solutions

Large User Communities

Most organizations face daunting obstacles in user management. Users within an organization often have far too many user accounts, with each system that controls sensitive material having a separate authentication procedure. This problem has been exacerbated by the growth in Web-based self-service applications – every other week, users have a new user account and password to remember. Organizations who want per-user data access and accountability do not want the administrative nightmare of managing users in each database or application users access. An organization opening its mission-critical systems to partners and customers does not want to create an account for each partner in each database the partner accesses; yet per-partner privilege and per-partner accountability is highly desired.

An increasing number of products view directories as the best mechanism to make enterprise information available to multiple different systems within an enterprise. The trend toward directories has been accelerated by the growth in use of the Lightweight Directory Access Protocol. These directories contain the user's identity information, as well as their roles and privileges to perform operations. Enterprise roles, roles that are defined across an enterprise and that apply to multiple applications, enable strong centralized user authorization. Also, an administrator can add capabilities to enterprise roles (granted to multiple users) without having to update each user's authorizations independently.

Storing this information in a central repository allows the administrator to grant and remove privileges that impact all of the organizational resources. Directory information that specifies users' privileges or access attributes is sensitive since unauthorized modification of this information can result in unauthorized granting or denial of user privileges or access. A directory that maintains this organizational information must ensure that only authorized system security administrators can modify privileges or access directory information.

Secure Communications

Communication mechanisms must support both confidentiality and data integrity requirements. It is important to secure the communications against network snooping

and data replay or modification (altering data on the wire or removing information during transmission). Network encryption, including both confidentiality and integrity algorithms, is a standard method for ensuring secure communications.

In the case of client-server Web-based applications, it is important to support encryption from the client browser to the middle-tier Web application server. Using Secure Sockets Layer Version 3 (SSL V3) has become accepted technology for this purpose. SSL provides authentication, integrity, and encryption services using public-key encryption.

It is also important to support encryption from the middle tier to the database. This can be provided through a variety of mechanisms, including both native encryption technology in database products and using SSL. Because different algorithms provide different features and assurance, it is important to support a variety of industry-standard encryption algorithms to protect the confidentiality of data: for example, the Data Encryption Standard (DES, see <www.itl.nist.gov/fipspubs/fip46-2.htm>); triple DES (see <http://csrc.nist.gov/cryptval/des.htm>); and RC4 (see <www.rsasecurity.com/rsalabs/faq/3-6-3.html>). Also, use integrity algorithms to verify that data have not been modified, including Secure Hash Algorithm (SHA)-1 (see <http://csrc.nist.gov/cryptval/shs.html>) and MD5 (see <www.rsasecurity.com/rsalabs/faq/3-6-6.html>).

The Federal Information Processing Standard (FIPS) 140-1, Security Requirements for Cryptographic Modules, was established to validate encryption products purchased by the U.S. government. Products are validated against FIPS 140-1 at security levels ranging from level one (lowest) through level four (highest). A FIPS validation ensures that the implementation of an encryption algorithm has been properly tested.

Auditing

A critical aspect of any security policy is maintaining a record of system activity to ensure that users are held accountable for their actions and that they do not abuse their privileges. Auditing implementations can and do vary by vendor; the following describes Oracle's auditing capabilities. Auditing options need to be highly granular to target the user actions of interest, to minimize the performance overhead of auditing, and to avoid analysis paralysis, in which there are too many auditing records to facilitate meaningful inspection. Ideally, audit records include enough granularity that an administrator can determine what the user requested as well as what was returned to the user at the time of the original request.

A robust database audit facility will allow organizations to audit database activity by statement, by use of system privilege, by object, or by user. One can also audit only successful or unsuccessful operations. For example, auditing unsuccessful SELECT statements may catch users on fishing expeditions for data they are not privileged to see. Database system logs that capture all changes to the database (required for recoverability of data) can be accessed for this purpose. The granularity and scope of these audit options allow customers to record and monitor specific database activity without incurring the performance overhead that more general auditing entails.

A needed auditing capability is one that enables organizations to define specific audit policies that can alert administrators to misuse of legitimate data access rights. What is desired is the ability to define audit policies, which specify the data access conditions that trigger the audit event and are tied to a flexible event handler to notify administrators (e.g., via a page) that the triggering event has occurred. An Oracle implementation of this feature captures the exact text of the statement the user executed in audit tables. In conjunction with other database features that reconstruct the result of a query at a past time, this auditing capability can be used to recreate the exact records returned to a user. A flashback or temporal query allows the recreation of the data a user accessed at the time of the original operation. This is an important feature for customers who have especially sensitive information they wish to share that requires strict accountability such as federal organizations selectively sharing information for counterterrorism purposes.

Many three-tier applications authenticate users to the middle tier, and then the transaction-processing monitor or application server connects as a super-privileged user and does all activity on behalf of all users. The user on whose behalf the middle tier is operating needs to be known to the database system. This allows the database to authenticate the real client, enforce access control based on least privilege, and audit actions taken on behalf of the user by the middle tier. To provide full accountability, the audit records should capture both the logged-in user (e.g., the middle tier) who initiated the connection and the user on whose behalf an action is taken. Auditing user activity, whether users are connected through a middle tier or directly to the data server, enhances user accountability and thus the overall security of multitier systems.

Need-to-Know Protection

The U.S. armed forces, as with many military and intelligence organizations, has a requirement to separate unclassified (but sensitive) information from classified information and to compartmentalize access to classified information. This includes the ability to limit data access based on an arbitrary, hierarchical data level (e.g., Secret, Top Secret), compartments (e.g., Project X), and control of its release (e.g., Releasable to United Kingdom).

During the 1990s, many vendors delivered products that provided multilevel security (MLS): the ability to enforce mandatory access control based on the comparison of a user's clearance to a label on the data. For example, a database containing products required for a joint military exercise with coalition partners could contain data that were viewable only by the United States, particular coalition partners, or all parties. The database would both separate the data and control access based on user clearance. However, MLS systems had a very low rate of adoption even among the user communities (DoD) who had the requirement for such systems.

At the same time, MLS systems were failing to be adopted by the markets that had demanded them; commercial organizations were taking advantage of the accessibility of the Internet to become e-businesses. Many commercial companies that wanted to open mission-critical systems to partners and customers over the Internet had an increased requirement for granular access control to the user or customer.

Companies offering application-hosting services also faced unique security challenges such as keeping data from different hosted user communities separate. The simplest way of doing this is to create physically separate systems for each hosted community; the disadvantage of this approach is that it requires a separate computer with separately installed, managed, and configured software for each hosted user community, providing little economies of scale to a hosting company. Business-to-business exchanges also faced requirements for both data separation and data sharing.

To address both the Internet requirements for data separation and data sharing and government requirements for granular access control, Oracle introduced the abil-

ity to provide programmable row-level access control. This capability called Virtual Private Database (VPD) is server-enforced, fine-grained access control together with a secure application context, enabling multiple customers and partners to have secure direct access to mission-critical data. VPD enables, within a single database, per-user or per-customer data access with the assurance that enforcement is not able to be bypassed. The result is lower cost of ownership in deploying applications since security can be built once in the data server rather than in each application that accesses the data. Security is stronger because it is enforced by the database: No matter how a user accesses data, security policies cannot be bypassed.

VPD can be built upon to support specific application policies. One such policy implementation developed by Oracle addresses the DoD and intelligence community requirement to automatically provide labeled data management and enforce label-based and compartmentalized data access. This policy implementation allows organizations to assign sensitivity labels to information, control access to that data based on those labels, and ensure that data are marked with the appropriate sensitivity label. For example, a counterterrorism application may separate data for "need-to-know" purposes based on selected agencies or groups within agencies (e.g., Secret: CIA, Defense Intelligence Agency). The ability to natively manage labeled data is a tremendous advantage for organizations managing data of different sensitivity levels by being able to provide the right information to the right people at the right level of secure data access.

Standards

The existence of and adherence to standards enable stronger security of an integrated system. Security standards are especially important since security generally needs to be integrated to work; there are very few security bolt-ons that can enhance or enable security that do not already exist in the underlying components. Security standards also facilitate the secure integration of disparate technology components.

Standards also usually result in a lower cost of ownership because integration costs are lower (more things work together out of the box), and component costs are generally lower if the products are differentiated on something other than proprietary, lock-in technology. In general, the price is high, and the quality (especially security, which tends to be costly to build) is lower in a monopoly or near-monopoly market.

It is especially important for the DoD to ensure interoperability between entities within one of the armed forces, among the various armed forces, and with coalition partners' systems.

Another benefit of standards is that there tends to be less security by obscurity; the security mechanisms, if they comply with a standard, are well-known rather than hidden and can also be certified or evaluated against the standard, thus providing consumers with confidence in the security of the resulting products.

Standards can include two types of technical interfaces. The first is the Public Key Certificate Standards (PKCS, see <www.rsasecurity.com/rsalabs/pkcs>), and Public Key Infrastructure X.509 (PKIX, see <www.ietf.org/html.charters/pkix-charter.html>). Second, are the independent measures of assurance for security components such as FIPS-140, which speaks to cryptographic module validation and the international Common Criteria, which is a formal security evaluation standard.

"... a counterterrorism application may separate data for 'need-to-know' purposes based on selected agencies or groups within agencies ..."

Public Key Infrastructure

In an effort to provide more secure computing environments, many customers are pursuing rapid adoption and deployment of public key encryption technologies. Public key infrastructure (PKI) describes the application of public key technologies to a computing infrastructure to ensure data privacy and to protect systems from unauthorized access.

PKI itself is a basic encryption technique that has many important applications for secure systems. One application is SSL. For example, a secure implementation of SSL requires at least a server-side certificate attesting to the identity of the server with which a user is attempting to connect. PKI can also enable strong client authentication, provided that the PKI credentials themselves are securely contained and accessed (for example, via a smart card).

PKI is also very scalable technology; in theory, a user who has been given a set of PKI credentials attesting to his/her digital identity can authenticate to servers and systems that he/she has never connected to before because the user's identity can be validated through PKI mechanisms. While this portability of credentials has limited practical application in many commercial organizations (realistically, Bank A will not accept user credentials issued by Bank B), the applicability within DoD is much more apparent. Each service member has precisely one identity as far as the U.S. armed forces is concerned. Once credentials are issued for that identity, they can be reused in multiple DoD systems.

The following are features that are important in database products to support PKI Infrastructure implementations:

- Client-based authentication using X.509 certificates stored in PKCS No. 12 containers. PKCS No. 12, titled Personal Information Exchange Format, specifies a standard for the transfer of identity information including private keys, certificates, etc.
- Wallets (PKI credential container) interoperable with third-party applications and portability across operating systems.
- Support for multiple certificates for each wallet, including Secure Multipurpose Internet Mail Extensions (S/MIME, see <www.rsasecurity.com/standards/smime>) signing certificates, S/MIME encryption certificates, and code-signing certificates.
- Client authentication using SSL.

Since PKI at best provides a single set of credentials, but not single sign on, it is helpful to use PKI in conjunction with single sign-on services as described below.

Single Sign On

In client-server database applications, strong authentication, and single sign on (SSO) are important features. A variety of different user authentication mechanisms are used depending on the application requirements. These can include user passwords, smart cards, token cards, and biometric authentication devices. SSO is provided by technologies such as Kerberos (see <http://web.mit.edu/kerberos/www>), Distributed Computing Environment (see <www.osf.org/dce>), and SSL.

Web-based SSO encompasses a different set of security issues than client-server SSO due to the stateless nature of Web-based connections. One approach to dealing with this problem is to use a central-

ized server to authenticate and pass authenticated identity securely to partner applications. With this approach, a log-in page is displayed to the user requesting a username and password. Once the user has done so, his/her password is verified and an SSO cookie is set in the user's browser. The client sends this encrypted cookie along with all subsequent HTTP interactions to the partner applications, authenticating the client with the centralized authentication server and avoiding the need for the client to re-authenticate as long as the cookie is valid.

Coupling this authentication technology with PKI allows a user to strongly authenticate to the centralized authentication server via SSL while obtaining the benefit of Web-based SSO after PKI-based authentication.

Availability

Databases and the Internet have enabled worldwide collaboration and information sharing by extending the reach of database applications throughout organizations and communities. This reach further highlights the importance of high availability in data management solutions. Small businesses and global enterprises alike – let alone the U.S. armed forces with their obvious need for high availability to support national security missions – have users all over the world requiring access to data 24 hours per day. Data availability includes the capacity to recover from unplanned outages, allow planned database maintenance while the database is in production and available to users, improve system manageability and serviceability, and provide enterprise-class disaster planning. Highly available solutions have three basic characteristics:

- Reliability: Reliable solutions are made of components that seldom fail.
- Recoverability: In the event a component does fail, a highly available solution quickly recovers without human intervention.
- Continuous Operation: Highly available solutions continue to provide service, even during maintenance activities.

Each component in a system should be designed to provide high availability, which means each component is reliable. In addition, each component must be able to recover from failures of supporting components in the stack. The frequency of failures and the speed of recovery determine the amount of unplanned downtime and application experiences. However, unplanned downtime is not the complete story. Each component must be able to provide continuous operation to meet an acceptable planned downtime target. This may require designing and building the system so that preventative maintenance can be performed while the application is online and users are accessing data. It is also important to plan for unforeseen incidents such as earthquakes and power outages that may prevent recovery for an extended period of time.

One of the true challenges in designing a highly available solution is examining and addressing all the possible causes of downtime. It is important to consider causes of both unplanned and planned downtime, including middleware, application, and network failures. Unplanned downtime can include component failure; hardware failures include system, peripheral, network, and power failures. Human error, a leading cause of failures, includes errors by an operator, user, database administrator, or system administrator. Another type of human error that can cause unplanned downtime is sabotage. The final category is disasters. Although infrequent, these causes of downtime can have extreme impacts on enterprises because of their prolonged effect on operations. Possible causes of disasters include fires, floods, earthquakes, power failures, and bombings. A well-designed, high-availability solution will account for all these factors in preventing unplanned downtime. Planned downtime can be just as disruptive to operations, especially in the DoD, which must support users in multiple time zones up to 24 hours per day. In these cases, it is important to design a system to minimize planned interruptions.

Databases systems are designed to address the causes of unplanned and planned downtime. In the event of a failure, a database can quickly and automatically recover. No committed data are lost. In addition, database systems support features that obviate the need for planned downtime, allowing administrators to perform many management and maintenance tasks while the system is online and data are fully accessible. Management tools are available that identify potential problems and rectify them before they affect data availability.

Assurance

It is important not only to support security features but also to have validation that the features have been implemented in a correct and secure manner. This is provided by formal and independent security evaluations. A commitment to past and continuing product evaluation of new releases against the Common Criteria (ISO-15048) and encryption technology against FIPS 140-1 is a proven measure of a product vendor's commitment to security. This level of commitment should be a requirement for use by the most security-conscious customers in the world: governments, defense, and intelligence agencies. The database, however, is only part of an enterprise-wide, end-to-end security model. A comprehensive approach to security, i.e., a multitiered distributed enterprise, is important to satisfying the mission of large government customers like the U.S. armed forces.

Conclusion

The DoD has a requirement for secure, interoperable, and available systems. While additional technical research and advancement will improve available security technology, much of the security technology needed to meet the DoD security requirements exists today. What is required is a commitment to use the security technology that exists, to demand secure and independently evaluated solutions, and to incorporate security into the entire computing infrastructure.◆

About the Author

Kevin J. Fitzgerald is senior vice president, Oracle Corporation Government, Education and Healthcare. He has been a leader in information technology sales and sales management for more than 25 years. Fitzgerald previously worked for Oracle from 1987-97 as vice president and general manager of the public sector sales group. During his initial tenure, he served in a number of sales roles and also helped initiate new product development areas and product enhancements, enabling Oracle to meet specialized government requirements. Fitzgerald served in the U.S. Air Force and has a bachelor's degree from Boston College.

Point of Contact: Tracy Strelser
1910 Oracle Way
Reston, VA 20190
Phone: (703) 364-6118
Fax: (703) 364-3026
E-mail: tracy.strelser@oracle.com

What is Software Quality Assurance?

Dr. Linda H. Rosenberg
NASA

Thursday, 2 May 2002
Track 7: 9:00 - 9:40
Room 251 A - C

Software directly impacts not only mission success but also mission safety. Software Quality Assurance (SQA) is critical to the success of every mission at NASA, but the roles and responsibilities are often misunderstood. SQA covers all phases of the software development process, including safety, reliability, independent verification and validation, and metrics. The purpose of this article is to help the reader understand software quality assurance.

Within the complex systems developed throughout the aerospace industry, software is playing an increasingly important role in mission success. Methods for developing and assuring software are often not well understood by program managers and, thus, are often simply ignored. In such a case, ignorance is far from bliss; it is dangerous. During the past few years, NASA has emphasized the faster, better, and cheaper approach to developing missions, thereby making it more important than ever to ensure the quality of its software products. It is this imperative that makes the role of Software Quality Assurance (SQA) critical in the short term, but also linked to mission success in the long term.

Assuring software quality requires that engineering knowledge and discipline be applied at all phases of the development life cycle. And just as with hardware, the final step in developing quality products culminates in rigorous testing before release. Quality assurance engineers are also required to possess sufficient domain knowledge to evaluate the completeness and correctness of system requirements, and they must have the ability to determine whether the design has incorporated all requirements accurately. Ultimately, these specialists are responsible for advising management when or whether a product is reliable and meets quality standards.

This article starts by discussing what is meant by SQA. It then discusses the aspects of how software quality assurance is applied to both the products and the process. The article continues with some of the major components of software assurance. Software metrics are used to help numerically determine the quality of the products, noting they are underutilized and often poorly understood. Another area of quality assurance not well understood is independent verification and validation (IV&V); this article will touch briefly on the role it plays in quality assurance. Finally, it discusses the ways in which software safety and reliability are assessed from a quality perspective. These two areas are often neglected despite their critical role in mission success.

Definitions

Software quality assurance is a combination of three concepts: quality, software quality, and software quality assurance.

> *"In the real work of software development, criteria for quality are identified and applied to differing extents as a result of trade-off decisions."*

While the terms are often used interchangeably, we need to understand the basics of quality before we can understand the components and problems of software quality assurance.

Quality Defined

Before defining software quality, we need to define what is meant by quality. The Institute of Electrical and Electronics Engineers' (IEEE) *Standard Glossary of Software Engineering Terminology* defines quality as "the degree to which a system, component, or process meets (1) specified requirements, and (2) customer or user needs or expectations [1]." The International Standards Organization (ISO) defines quality as "the totality of features and characteristics of a product or service that bear on its ability to satisfy specified or implied needs [2]." IEEE and ISO definitions associate quality with the ability of the product or service to fulfill its function. This is achieved through the features and characteristics of the product.

While this definition seems to be clear and unambiguous, the concept of quality really is not. Kitchenham states quality is "hard to define, impossible to measure, easy to recognize [3]." Gillies states that "Quality is generally transparent when present, but easily recognized in its absence [4]."

Therefore, while we can define quality in theory, in practice, and in use, an absolute definition is elusive.

Software Quality Defined

Software quality is defined in the *Handbook of Software Quality Assurance* in multiple ways but concludes with this definition: "Software quality is the fitness for use of the software product [5]." This definition implies the evaluation of software quality related to the specification and application of software quality. There are, however, criteria that help in the evaluation of software quality. For each project, the appropriate criteria need to be identified for the environment.

Two of the most often-cited models applying the criteria are the GE model proposed by McCall, which was later adapted by Watts, and the Boehm model [4]. Below is a combined list of definitions of quality criteria for software.

- Correctness: extent to which a program fulfills its specifications.
- Efficiency: use of resources execution and storage.
- Flexibility: ease of making changes required by changes in the operating environment.
- Integrity: protection of the program from unauthorized access.
- Interoperability: effort required to couple the system to another system.
- Maintainability: effort required to locate and fix a fault in the program within its operating environment.

- Portability: effort required to transfer a program from one environment to another.
- Reliability: ability not to fail.
- Reusability: ease of re-using software in a different context.
- Testability: ease of testing the program to ensure that it is error-free and meets its specification.
- Usability: ease of use of the software.

In a perfect world all of these criteria would be met, but software is not developed or run in such a world, and trade-offs are a part of all development projects. Often the most efficient software is not portable, as portability would require additional code, decreasing the efficiency. Usability is subjective and varies depending on the system users. When using the above criteria to define the assurance objectives of a software system, the purpose and use of the system must be taken into account. In the real work of software development, criteria for quality are identified and applied to differing extents as a result of trade-off decisions.

Software Quality Assurance Defined

Again referencing IEEE, quality assurance is defined as "a planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements [1]." This definition needs to be adapted to software taking into account that, unlike hardware systems, software is not subject to wear or physical breakage; consequently, its usefulness over time remains unchanged from its condition at delivery. Software quality assurance must be a systematic effort to improve the delivery condition. In the *Handbook of Software Quality Assurance*, the following definition is given: "Software quality assurance is the set of systematic activities providing evidence of the ability of the software process to produce a software product that is fit to use [5]." These activities are evaluated in part against the above criteria and measured as described in a later section of this article.

Software Quality Assurance Applied

The focus, therefore, of SQA is to monitor continuously throughout the software development life cycle to ensure the quality of the delivered product. This requires monitoring both the processes and the products. In process assurance, SQA provides management with objective feedback regarding compliance to approved plans, procedures, standards, and analyses. Product assurance activities focus on the changing level of product quality within each phase of the life cycle, such as the requirements, design, code, and test plan. The objective is to identify and eliminate defects throughout the life cycle as early as possible, thus reducing test and maintenance costs.

Process Assurance

It has been proven that the use of standards and process models has a positive impact on the quality of the final software. The purpose of standardization of SQA ensures that there is discipline and control in the software development process via independent evaluation [5]. ISO 9000 provided a way to gain external accreditation for a quality management system. Many companies have used the application of ISO to software, but the complaint is that it tends to fossilize procedures rather than encourage process improvement [4].

One of the most common software development models is the Software Engineering Institute's Capability Maturity Model® (CMM®), which has recently developed into the CMM Integration℠ (CMMI℠). The basic premise underlying both CMM and CMMI is that the quality of the software product is largely determined by the quality of the software development and maintenance processes used to build it [6].

> *"It has been proven that the use of standards and process models has a positive impact on the quality of the final software."*

Many commercial standards are also found in common practice for software development. Many organizations such as The Department of Defense and NASA have, in the past, developed their own standards for software development, but recently have embraced the use of commercial standards instead. It is now NASA's policy to use commercial standards whenever possible, thus encouraging more standardization not only across NASA but within industry also.

Product Assurance

At NASA's Goddard Space Flight Center (GSFC), software quality assurance is carried out by an independent group of people whose sole function is to monitor quality implementation. The Assurance Management Office recently created a list of tasks that SQA should perform during each phase of the software development life cycle. This list is comprehensive and starts in the concept phase of a proposed project and concludes with the operations and maintenance phase. For example, in the concept phase, SQA should generate and/or assist in the development/review of various program/project plans, including but not limited to project management plans, subcontract management, etc. In the requirements phase, SQA should obviously generate and/or assist in the generation of requirements, but it should also do activities such as observing, witnessing and/or participating in prototyping activities.

To accomplish all of these tasks would be an ideal set of SQA activities on a project, but projects rarely have sufficient funds or need to perform them all. For most projects, the amount of SQA to be applied is negotiated based on the purpose, degree of mission risk, and the funding level of the project. This negotiation is critical to the success of SQA. In the following sections, I will discuss four activities in which SQA must participate during all phases: metrics, IV&V, safety, and reliability.

Metrics

Software metrics are often ignored during the early software development life-cycle phases and are not an activity generally associated with SQA – but should be. For SQA practitioners, with their responsibility for assuring both the processes and products of the software development, measurement is critical. Throughout each of the life-cycle phases, metrics can be used to help in the evaluation.

The Software Assurance Technology Center (SATC) at GSFC has identified relevant metrics that can help projects better evaluate the quality of their products at fixed points within their development. For example, SATC developed a tool that derives metric information by analyzing requirement specification documents. Known as Automated Requirements Measurement,[1] this tool provides indicators of the quality of the requirements set. The tool's objective is to identify terms within the text that may cause

requirements to be ambiguous and hence difficult to test and to identify any requirements that are incomplete [7].

It is up to the SQA organization to be cognizant of available and relevant metrics that help evaluate and assure products. When projects consistently use software metrics as part of their development, the SQA team needs only to validate the metrics and ensure the correct data interpretation. If a project is not employing metrics, however, then it is the responsibility of SQA to encourage, and perhaps facilitate, their use or to develop an independent metrics program for sufficient insight into the development.

Independent Verification and Validation

IV&V is defined by three components; it must be independent technically, managerially, and financially. IV&V must prioritize its own efforts, identifying where to focus its activities. It must have a clear reporting route to the program management, and the budget for these efforts must be allocated and controlled by the program. Control must occur at a level that is independent of the development organization such that the effectiveness of the IV&V activity is not compromised.

Verification is defined as the process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase, i.e., whether or not it is internally complete, consistent, and correct enough to support the next phase. Validation is the process of evaluating software throughout its development process to ensure compliance with software requirements. Verification often asks the question, "Are we building the product right?" Validation asks, "Are we building the right product?"

NASA has a facility in West Virginia whose primary purpose is the accomplishment of IV&V. Without SQA, IV&V is expensive and often less effective. Where SQA is a broad blanket across the project, overseeing all process and product activities, including software, IV&V focuses on only those processes and products determined to have the highest risk and does an in-depth evaluation of them.

Safety

Safety is a team effort and is everyone's responsibility. Software is a vital part of the system. Project managers, systems engineers, software leads and engineers, software assurance or quality assurance (QA), and system safety personnel all play a part in creating a safe system. Safety-critical software is defined by the NASA Software Safety Standard as "Software that directly, or indirectly, contributes to the occurrence of a hazardous system state, controls or monitors safety critical functions, runs on the same system as safety critical software or impacts systems that run safety critical software, or handles safety critical data [8]." The goal is for the QA activity to ensure that software contributes to the safety and functionality of the whole system.

When a device or system could possibly lead to injury, death, or the loss of vital (and expensive) equipment, system safety is always involved. Often hardware devices are used to mitigate the hazard potential or to provide a *fail-safe* mechanism should the worst happen. As software becomes a larger part of electromechanical systems, hardware hazard controls are being replaced or backed up by software controls. Software has the ability not only to detect certain types of error conditions more quickly than hardware but also to respond more intelligently, thereby avoiding a potentially hazardous state. The increased reliance on software means that the safety and reliability of the software become vital components in a safe system [8].

"When projects consistently use software metrics as part of their development, the Software Quality Assurance team needs only to validate the metrics and ensure the correct data interpretation."

Reliability

IEEE defines software reliability as "The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system, as well as a function of the existence of faults in the software [9]." Using this definition, expectations of reliability must be based on how the system is to be used and for what length of time. At NASA, many of our satellites fly for multiple years; the reliability of their software must support the expected lifetime. The conditions of that software's use will be specified by the satellite's mission.

IEEE continues to define software reliability management as "The process of optimizing the reliability of software through a program that emphasizes software error prevention, fault protection and removal, and the use of measurements to maximize reliability in light of project constraints such as resources, schedule, and performance [9]." This definition puts the burden of reliability not just on the testing phase, but on the entire life cycle to ensure errors are prevented starting in the requirements phase determining the quality of such attributes as phrasing, completeness, and clarity. Throughout the life cycle, errors should be detected and removed using such techniques as code walkthroughs and inspections. Relevant measurements should be used at all phases to ensure the effectiveness of all assurance activities. In the testing phase, reliability can be evaluated using one of the many reliability models. These models, however, must be applied with very strict rigor to ensure accuracy.

It is the responsibility of the SQA organization to ensure that reliability is continuously promoted and evaluated throughout the life cycle as specified above. Quality cannot be tested in at the end of a project; it must be built in as the software is being developed. Reliability also impacts safety – a system cannot be deemed safe if it is not reliable.

Conclusion

SQA is faced with many challenges starting with the method of defining quality for software. There needs to be a common understanding as to what is high-quality software, but the software usage environment usually influences the final definition. There are many aspects of SQA, from those within the phases of the software development life cycle to those that span multiple phases, i.e., safety, reliability, and IV&V. SQA is a very complex area that is critical to the ultimate success of a project; it is also one that requires a rather diverse set of skills. New knowledge areas such as software safety and reliability are now being added to the core set of required skills. Finally, SQA must be independent from development organizations to be successful.◆

References

1. IEEE Std 610.12-1990. Glossary of Software Engineering Terminology. Institute of Electrical and Electronics

Engineers, Inc., 1990.
2. ISO 9003-3-1991. Quality Management and Quality Assurance Standards, Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software. International Standards Organization, 1991.
3. Kitchenham, Barbara, and Shari Lawrence Pfleeger. "Software Quality: The Elusive Target." IEEE Software 13, 1, Jan. 1996: 12-21.
4. Gillies, Alan C. Software Quality, Theory and Management. International Thomson Computer Press, 1997.
5. Schulmeyer, G. Gordon, and James I. McManus. Handbook of Software Quality Assurance, 3rd ed. Prentice Hall PRT, 1998.
6. Software Engineering Institute. Capability Maturity Model. Carnegie Mellon University, 1991.
7. Wilson, W., L. Rosenberg, and L. Hyatt. "Automated Quality Analysis of Natural Language Requirement Specifications." Proceedings of the 14th Annual Pacific Northwest Software Quality Conference, Portland, Ore., 1996.
8. NASA-STD-8719.13A. NASA Software Safety Standard. NASA, 2001.
9. IEEE Std 982.2-1988. Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software. Institute of Electrical and Electronics Engineers, Inc., 1988.

Note

1. Available on the SATC Web site at no cost, see <http://satc.gsfc.nasa.gov>.

About the Author

Linda H. Rosenberg, Ph.D., serves as the chief scientist for Software Assurance for Goddard Space Flight Center, NASA. She is a recognized international expert in the areas of software assurance, software metrics, requirements, and reliability. Dr. Rosenberg has a doctorate degree in computer science, a master's of engineering science degree in computer science, and a bachelor's of science degree in mathematics.

Office of Systems Safety and Mission Assurance
Goddard Space Flight Center, NASA
Building 6 Code 300
Greenbelt, MD 20771
Phone: (301) 286-0087
Fax: (301) 286-1667
E-mail: linda.h.rosenberg@gsfc.nasa.gov

Surviving the Top 10 Challenges of Software Test Automation

Randall W. Rice
Rice Consulting Services, Inc.

Thursday, 2 May 2002
Track 8: 1:00 - 1:40
Room 251 D - F

Capture/playback tools make it possible to repeat two or more tests identically and compare the results. This article focuses on capture/playback tools, which hold the largest market share of any test tool category, and on examining trouble spots in test automation, dealing with them proactively and perhaps mitigating the risks of tool abandonment. The purpose of this article is to outline the 10 major challenges that the author sees most often in organizations struggling to make effective test automation a reality.

Capture/playback tools capture or record the actions performed during a test session into software-like scripts that can be replayed against the same or an updated version of the software, allowing a comparison of identical test results. A difference in test results may indicate the presence of a regression defect.

For the past six years, I have been asking training and conference audiences, "How many of your organizations own some type of automated capture/playback test tool?" Typically, 80 percent to 90 percent of the audience will raise their hands. However, the typical response drops to about 10 percent to 20 percent of that group when I next ask, "How many of you who just raised your hand would consider automated test tools an integral part of your testing effort?"

Obviously, there is a large gap between the people who own automated test tools and people who actually benefit from test automation. Also, my survey findings have not changed significantly during the past six years. The observation that automated test tools account for a lot of *shelfware* is not new or all that insightful. In fact, the shelfware problem is shared with many types of software tools.

When faced with the dynamic world of automated test tools, many organizations make the best choice they can and then try to make the tool work in their environment. They hope the rest of the organization will embrace the tool as well.

The purpose of this article is to outline the 10 major challenges that I see most often in organizations struggling to make effective test automation a reality. The earlier these challenges are understood, the better prepared an organization will be to deal with them. Plus, the more an organization understands the issues of test automation, the less risk it incurs in time and money when acquiring a tool.

These challenges are presented in order from the least to highest impact on the overall test automation effort. It is also important to understand that even organizations that have developed core competencies in working with test automation struggle at times with these challenges.

"Test automation promises increased productivity and accuracy, which is where the business case must be made. The cost of a single defect ... can offset the price of one or more tool licenses."

1. Lack of Tool Availability

The lack of tool availability is usually the result of the following:

1. The tool is available, but you cannot get the funding for it.
2. There does not seem to be a tool on the market that does what you need or fits in your environment.

The first issue is based in management's understanding of testing and the priority it is given in terms of funding. While the cost of automated test tools is high compared with more common software packages such as office automation products and software development suites, the anticipated value of the tools is in reduced time and greater testing precision. Test automation promises increased productivity and accuracy, which is where the business case must be made. The cost of a single defect in most organizations can offset the price of one or more tool licenses. It is all a matter of where management chooses to spend the money – in defect detection or post-production rework, which is many times more costly than early defect detection.

Not having tools available in a particular environment is more troublesome. Although there is now automated tool support in most environments, this does not mean the support in every environment is great. In older environments, tool support is very limited.

If funding is the issue, consider the following:

- Measure the current cost of defects, especially in post-implementation rework. Use this information to help build a case for faster and more reliable testing using tools.
- Show the value of automated test tools for other groups besides testers such as the value of developers using the tools.

If getting a good technical fit is the issue, consider the following:

- Network with other testers to find information about lesser-known test tools. Online quality assurance forums are also good places to query people about tools in lesser-supported environments.
- Try to find tools that will work between platforms. This will likely require the use of PC-based emulators as opposed to host-based tools.
- Investigate the possibility of building your own tools or, at least, achieving a level of test automation using common scripting commands and comparison programs.

2. Lack of Tool Compatibility and Interoperability

Tool incompatibility and lack of interoperability are seen in organizations that have diverse technologies and applications. The desire is to be able to auto-

mate tests that bridge applications and platforms. This is a big challenge because most automated test tools have proprietary scripting languages and approaches, and, for the most part, competing vendor tools are not interoperable.

A related challenge is to automate identical tests on a number of different platforms. This requires tool compatibility among various computing platforms and the ability to share scripts between tools and platforms.

If compatibility and interoperability are the issues, consider the following:

- Select tools that have cross-platform capability to the greatest extent possible.
- Consider writing shell scripts and bridging scripts, perhaps in non-proprietary scripting languages such as Tcl.
- Evaluate critically whether the ability to perform cross-platform testing is a firm requirement.

3. Lack of Configuration Management Processes

Test automation is using software to test software. This means that items created using the automated test tools should be subject to the same level of control as any other software asset.

When software configuration management (SCM) is not in place for automated testing, the discipline is missing to work with the tools. The following occurs without SCM for automated test tools:

- Effort is duplicated because different people may each be building similar test scripts.
- Reuse is not realized because people are all creating test scripts for single-use purposes.
- Existing automated test scripts are at risk of corruption if they are modified without the knowledge of the original author.

Here is what is required for effective SCM for test automation:

- A workable process that everyone using the tool can understand and follow.
- A tool to manage the ownership, versions, and organization of the automated test scripts.
- Someone to own the SCM process and ensure that people are following it.

Many of the popular automated test tools have integrated test case and test script management applications. However, you still need the process and the people to make the SCM effort work. You can also build your own test management tool using a database and basic file organization to group related tests into suites.

A related issue is keeping up with changes to applications that are under test. This has been one of the biggest challenges in test automation since its inception. The degree of difficulty in dealing with application changes in automated testing software depends on the tool and the technologies involved. In the object-based world, the more robust tools can be configured to ignore user interface changes as long as the objects still behave the same. However, if the tool uses row-and-column positioning, then each application change will require a change (or many changes) to the automated test scripts.

In character-based applications such as mainframe Customer Information Control System applications, all of the changes that impact the user interface will most likely require maintenance to the automated test scripts that test those interfaces.

"If you do not know which tests are the most important and which tests are the most applicable for automation, the tool will only help perform a bad test faster."

Here are solution strategies for this challenge:

- During your tool search, consider the people and processes that will be required to manage the automated test cases and test scripts.
- If you are in the object-based environment (such as graphical user interfaces), look for tools that accommodate changes to the user interface gracefully. These tools cost more than those that do not offer such flexibility. However, many people have found that the added cost is small compared with the cost on continued manual maintenance of the test software.
- Consider automated test scripts as part of an application's configuration set.
- Involve the prospective test automation SCM person in evaluating test tools and their respective test management offerings.
- Investigate the use of existing SCM tools currently owned by your organization.
- Trace your automated test scripts to functional requirements and defects.

4. Lack of a Basic Test Process or Understanding of What to Test

Most automated test tools do not tell you what to test. Even the tools that have test-case generation features do so at a user-interface level and not at the functional-requirements level.

If you do not know which tests are the most important and which tests are the most applicable for automation, the tool will only help perform a bad test faster. This is a universal principle that I often illustrate with the example of a power tool.

Let us say that I want to build a bookcase. I try cutting the wood with a handsaw, but it is far too slow and laborious. So, I decide to go to the hardware store and buy a power saw. After the purchase of the tool that I can afford, that looks good, or that the salesperson convinces me to buy, I go home and start cutting wood. However, if I do not have bookcase plans or a very good understanding of how to build a bookcase, the saw will just help me make my mistakes faster! To be successful, I will need to first learn enough woodworking skills to understand not only the "what" and "when," but the "why" and "how" of building the bookcase. Then, I'm ready to use the tool effectively.

The tool vendors can train you to use the tool with all of its functionality, but the burden is on you to examine your own applications and determine which functions should be tested and to what extent.

Here are solution strategies for this challenge:

- Create a set of evaluation criteria for functions that you will want to consider when using the automated test tool. These criteria may include the following:
  - Test repeatability.
  - Criticality/risk of applications.
  - Operational simplicity.
  - Ease of automation.
  - Level of documentation of the function (requirements, etc.).
- Examine your existing set of test cases and test scripts to see which ones are

most applicable for test automation.
- Examine your current testing process and determine where it needs to be adjusted for using automated test tools.
- Be prepared to make changes in the current ways you perform testing.
- Involve people who will be using the tool to help design the automated testing process.
- Train people in basic test-planning skills.

5. Lack of Tool Ownership and Acceptance

The challenge with lack of tool ownership and acceptance is that the tool is not applied or is ignored. This is often the result of someone's good intention of buying a tool to make life easier, but the rest of the people do not use it. The following are some of the reasons for lack of tool ownership and acceptance:
- Difficulty using the tool.
- Not enough time to learn the tool and still perform normal work levels.
- Lack of tool training.
- Lack of management support for tool use.
- Lack of tool support, either internally or from the vendor.
- Tool obsolescence.

Here are solution strategies for this challenge:
- Do not cut the tool training. Training does not guarantee success, but without it you are at risk of tool abandonment.
- Have someone in your organization in the role of a *tool smith*. This person's job is to be the resident expert on the tools used for testing.
- Management needs to emphasize that the tool effort is important to them, and that tool usage is a required part of the testing process.

6. Inadequate Tool Training

We discussed the training issue previously in "Lack of Tool Ownership and Acceptance," but this challenge carries its own set of concerns. Some of the key issues are as follows:
- Skipping the vendor's training. The main motivation for this is lack of time and/or money. You will spend more of both without the training!
- Not getting the right training due to the incorrect selection of topics. For example, some tool users will need to learn in detail the tool's test scripting language, while other users will need to learn only the basic tool functionality.
- Inability to apply the training to your environment. This is where you learn to use the tool on the vendor's canned example but have difficulty getting the tool to work on your own applications.
- Trying to learn by self-study. Yes, it can be done, but it takes time and dedication. More often than not, people tend to spend time with only the basic functions and do not gain the benefit of learning the lesser known and perhaps more powerful tool features.
- Not enough time for training. This goes along with the "dive-right-in" approach often seen in information technology groups. When time is scarce, people tend to gravitate toward the easy and basic functions at the expense of not learning the more difficult but more powerful ones.

"Perhaps the greatest challenge seen in management support is balancing high expectations of tool benefits against the time, effort, and discipline it takes to implement the tool."

Here are solution strategies for this challenge:
- Include money in the tool proposal for training at least a core group of people.
- Match people to the most applicable training topics.
- Have tool training performed by the vendor at your location using some of your own applications as exercises.
- Find a skilled local consultant experienced with the tool to sit with your team for about three to four weeks to help get you started in creating automated tests. It is very important that your team does most of the work to accomplish the transfer of knowledge!

7. Incomplete Coverage of Test Types

As you profile your tests and defect types, you will often find a wide variety of test types that need to be performed. These include tests for the following:
- Correctness.
- Reliability.
- Security.
- Performance.
- Usability.
- Interoperability.
- Compatibility.
- Data Conversion.

Although the tool may be very adept at automating many of these tests, there may be test types that the tool simply cannot support. In fact, most organizations are very happy with a coverage level of 80 percent of their existing test case libraries.

Here are solution strategies for this challenge:
- During tool evaluation, prioritize which test types are the most critical to your success and judge the candidate tools on those criteria.
- Understand the tools and their tradeoffs. You may need to use a multi-tool solution to get higher levels of test-type coverage. For example, you will need to combine the capture/playback tool with a load-test tool to cover your performance test cases.
- Manage expectations by reminding people that 100 percent test type coverage is not likely. However, by automating 80 percent of the tests, you have time to deal with the rest manually.

8. Lack of Management Support

Management support is needed in designing and deploying test processes that will support the effective use of test tools, reinforce the role and use of automated test tools in the organization, and allow time for tools to be integrated in the testing process.

Without management support, the entire test automation effort is at risk. If management does not clearly and consistently show their support for test automation, people will be less inclined to show interest in using the tools. This is a major concern, especially considering that overcoming the learning curve of some tools requires dedication.

Perhaps the greatest challenge seen in management support is balancing high expectations of tool benefits against the time, effort, and discipline it takes to implement the tool. Management may become impatient about the lack of tool progress and shift their support to other initiatives.

The pressure is on the people who

made the business case for the tools to show progress in a given timeframe. The problem is there are many unforeseen things that can delay or derail a tool initiative. In reality, if people fully knew all of the future problems with any given effort, they would be very reluctant to proceed. While there is a place for optimism in acquiring tools, a heavy dose of realism is also needed to keep expectations in line with what is achievable.

Here are solution strategies for this challenge:

- Communicate that it takes time and planning to build a firm foundation of people, processes, and the right tools.
- When making the case to management for acquiring test tools, present the challenges as well as the benefits.
- Reinforce to management that they carry a great deal of influence in how people will accept automated test tools.
- Keep management informed of tool progress and issues that arise.

9. Inadequate Test Team Organization

Most test organizations learn that automated testing is a new world in terms of how tests are designed and maintained. Most tests require more than just capture/playback. The tool user must also be able to work with the tool's scripting language to accurately replay the test session. It helps if the tool user is comfortable working with coding languages, otherwise, there is a risk that the tool will not be used.

Here are solution strategies for this challenge:

- Add a person to the test team who is a *test scriptor*. This person should be comfortable in working with code and be able to take the basic test that has been designed by a test analyst and convert it into an automated script.
- Start simple with basic scripting concepts and add complexity later.

10. Buying the Wrong Tool

Buying the wrong tool is listed as the No. 1 challenge in test automation because no matter what kind of process or organization you have, if the tool is not a good technical or business fit, people will not be able to apply it.

We know that a good process and organization are also essential for test automation. However, if the tool will not function at a basic level, people using the tool will simply give up trying to use it.

Unfortunately, too few people do adequate research before buying a test tool. Adequate research includes defining a set of tool requirements based on what the intended users of the tool need to accomplish, developing a set of evaluation criteria by which candidate tools will be judged, and taking the experience of other people who have used the tools under consideration.

Here are solution strategies for this challenge:

- Take time to define the tool requirements in terms of technology, process, applications, people skills, and organization.
- Involve potential users in the definition of tool requirements and evaluation criteria.

"Adequate research includes defining a set of tool requirements based on what the intended users of the tool need to accomplish, developing a set of evaluation criteria by which candidate tools will be judged, and taking the experience of other people who have used the tools under consideration."

- Build an evaluation scorecard to compare each tool's performance against a common set of criteria. Rank the criteria in terms of relative importance to the organization.
- Perform a proof of concept (POC) as opposed to an evaluation. In a POC, the vendor often sends their technical team to your site to automate tests using your applications in your environment. Usually, a POC takes about one day to perform. The planning of the POC should be based on the evaluation scorecard. Testers should identify and define the most critical and most common tests they currently perform manually. These tests are often the ones that consume the most time and are the ones that offer the highest payback in test automation.

Summary

These 10 challenges are certainly not the only ones that are seen in test automation, but they are very common and have been the cause for many test automation project failures.

Successful software test automation is possible if fundamental issues are addressed and managed. Success depends on multiple factors that require the coordination of efforts between various groups in an organization. Automated software testing is truly a different way of testing and requires adjustments to current test methods and organizational structures. However, the payback from test automation can far outweigh the costs.◆

Additional Reading

1. Perry, William E., and Randall W. Rice. Surviving the Top Ten Challenges of Software Testing. Dorset House Publishing, Mar. 1998.
2. Fewster, Mark, and Dorothy Graham. Software Test Automation. Addison Wesley Longman, May 2000.
3. Dustin, Elfriede, Jeff Rashka, and John Paul. Automated Software Testing. Addison Wesley Longman, June 1999.

About the Author

Randall W. Rice is a leading author, speaker, and consultant in the field of software testing and software quality. Rice, a certified quality analyst and certified software test engineer, has worked with organizations worldwide to improve the quality of their information systems and automate their testing processes. Rice has more than 25 years experience building and testing mission-critical projects in a variety of environments, including defense and private sector projects.

Rice Consulting Services, Inc.
P.O. Box 891284
Oklahoma City, OK 73189
Phone: (405) 793-7449
Fax: (405) 793-7454
E-mail: rrice@riceconsulting.com

Information Security System Rating and Ranking

Dr. Rayford B. Vaughn Jr., Ambareen Sira, and Dr. David A. Dampier
Mississippi State University

Wednesday, 1 May 2002
Track 5: 9:00 - 9:40
Room 250 A - C

The term assurance has been used for decades in trusted system development to express the notion of confidence in the strength of a specific system or system of systems. The unsolved problem that security engineers must struggle with is the adoption of measures or metrics that can reliably depict the assurance associated with a specific hardware and software architecture. This article reports on a recent attempt to focus needs in this area and suggests various categories of information assurance metrics that may be helpful to an organization that is deciding which set is useful for a specific application.[1]

We believe that the provision of security in systems is a subset of the systems engineering discipline, and that it has a heavy software-engineering component. As software engineers, we understand that the determination and application of measures and metrics is not an exact science, nor is it easily accomplished. We also realize that this difficulty carries over to the trusted systems world. How one measures the degree of protection present is, today, an unsolved question and is primarily accomplished by craftsmanship and not science.

This issue of rating and ranking systems in terms of their assurance characteristics was at least partially addressed at a workshop on information security system ratings and ranking in Williamsburg, Va.,[2] in spring 2001. We will hereafter refer to this as *the workshop*, as we reference it in support of our belief.

Workshop Findings and Observations

It appears that while we often claim to have metrics that prove or indicate assurance levels, we do not seem to be able to prove that correctness, maintainability, reliability, and other such nonfunctional system requirements are in the software we build. We also tend to use empirical evidence based on historical performance data in claiming system strength. However, knowing that a particular defensive strategy has worked well in the past for an organization really says very little about its strength for the future. Examples of software engineering difficulties that we face in predicting a system's strength include the following:

- Software is not subject to the laws of physics. In most cases, we cannot apply mathematics to code to prove correctness in the same way a bridge builder can apply formulae to prove structural strength characteristics.
- People who are, by nature, error prone build software. In the end, any one of them can intentionally or unintentionally corrupt the system and greatly diminish assurance.
- Compositions of mechanisms used to construct a security perimeter comply with no known algebra. We remain reliant on the expertise of our systems administrators or security engineers.

> *"... while we often claim to have metrics that prove or indicate assurance levels, we do not seem to be able to prove that correctness, maintainability, reliability, and other such nonfunctional system requirements are in the software we build."*

- It is easier to attack a system today (an assurance issue) than it was years ago. This trend is likely to continue as attack tools are further automated, shared, and explored on a global basis.

The workshop attempted to address these issues and others. Although many specific techniques and suggestions were proffered to the group, it was apparent to all that some combination of measures was essential, and that this combination could not generically be applied across all interest domains. Similarly, it was clear that the measures or metrics adopted by an organization to determine assurance need to be frequently revisited and re-validated.

Attempts to apply a single rating to a system have been tried in the past and have failed [1, 2]. The workshop organizers also agreed that the problem domain might be best viewed using a non-disjoint partitioning into technical, organizational, and operational categories.

Definitions agreed upon by the conference organizers in the technical category were measures/metrics that are used to describe and/or compare technical objects (e.g., algorithms, products, or designs). Organizational measures might be used with respect to processes and programs. Operational measures are thought to describe *as is* systems, operating practices, and specific environments.

An interesting characterization of information security metrics came from Deb Bodeau of The Mitre Corporation [3] who pointed out that a proper view of these metrics might be a cross product involving what needs to be measured, why you need to measure it, and for whom you are measuring. Her characterization of this view in Figure 1 is enlightening.

Another interesting observation made by several attendees was that the desired purpose for such measures and metrics seemed to vary between the government and commercial sectors. Government applications seem much more likely to use metrics and measures for upward reporting. Answering such questions as "What is our current assurance posture?" "How are we doing this month compared with last?" and "Are we compliant with applicable regulations and directives?" seemed to be a driver for the metrics needed by government.

The representatives at the workshop from the commercial world seemed less interested in these questions and more inclined to look for answers to the ques-

tions "How strong is my security perimeter?" "What is the return on my investment?" "What is my level of risk or exposure?" and "How does product performance compare?" The commercial sector seemed to have far more interest in technical and operational measures than in process or organizational measures.

The workshop attendees had hoped to find a number of objective, quantitative metrics that could be applied. Although unanimous agreement was not reached, it was apparent to most that such metrics were in short supply, had to be combined with other measures or metrics in a particular context, and were generally not very useful on their own. Many more measures that would be considered subjective and/or qualitative appeared more useful.

Examples of more useful measures might include adversary work factor – a form of penetration testing. An excellent discussion of this topic is found in Schuedel and Wood [4]. Although penetration techniques are not truly repeatable and consistent, the workshop attendees agreed that their results were meaningful and useful. Risk assessments, in their various forms, were also found to be useful measures of assurance. Such assessments are accomplished in a variety of ways, but tend to focus attention in the proper areas and give a good indication of how one is postured to withstand attacks on a system.

Information Assurance (IA) metrics are essential for measuring the *goodness* of IA, and we believe that overall useful IA metrics are possible. There is general agreement that no single system metric or any *one perfect* set of IA metrics applies across all systems. Which set will be most useful for an organization largely depends on its IA goals; its technical, organizational, and operational needs; and the resources that it can make available.

In order to help an organization investigate options for IA metrics, it is useful to look at the different categories of IA metrics in general. These categories are described as follows:[3]

Objective/Subjective

Objective IA metrics (e.g., mean annual downtime for a system) are more desirable than subjective IA metrics (e.g., amount of training a user needs to have to securely use the system). Since subjectivity is inherent in IA, subjective IA metrics are more available.

Quantitative/Qualitative

Quantitative IA metrics (e.g., number of failed login attempts) are more preferable than qualitative IA metrics (e.g., the Federal Information Technology Security Assessment Framework [5] self-assessment levels).

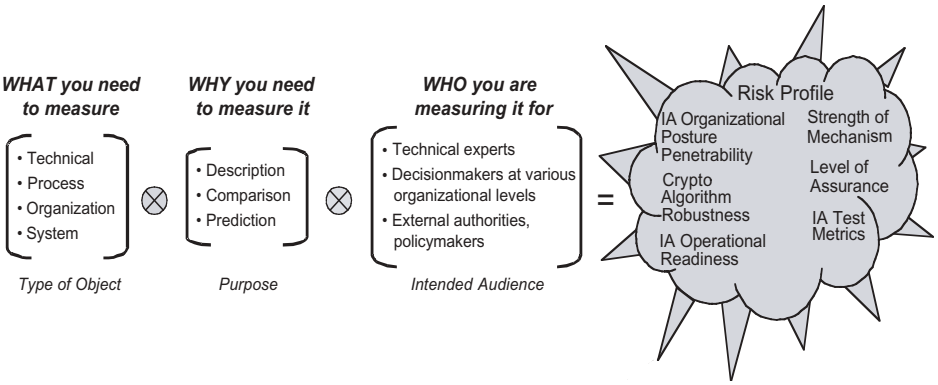


Figure 1: *Characterization of Information Security Metrics (Bodeau)*

Static/Dynamic

Dynamic IA metrics evolve with time, static metrics do not. An example of static IA metrics can be the percentage of staff who received annual security refresher training [3]. This metric can degrade in value if the content of the course does not change over time. An example of dynamic IA metrics can be the percentage of staff who received training on the current version of the software package they use. Most metrics used in penetration testing are dynamic. Dynamic IA metrics are more useful than static because best practices change over time with technology. There is always need to adapt metrics in compliance with best practices [6].

Absolute/Relative

Absolute metrics do not depend on any other measures, and these either do or do not exist [3]. For example, the number of systems administrator, networking, and security-certified security engineers in an organization is an absolute metric. Relative metrics are only meaningful in context. For example, the number of vulnerabilities in a system cannot assess the system's IA posture alone. The type and strength of vulnerabilities are also important in this context for making any decision about the system's IA posture. The majority of IA metrics are relative, and so they would not be good for use as a single-system metric.

Direct/Indirect

Direct IA metrics can be generated from observing the property that they measure. For example, the number of invalid packets rejected by a firewall over a certain period of time. Indirect IA metrics are derived by evaluation (e.g., ISO Standard 15408 The Common Criteria) and/or assessment (e.g., risk assessment). Although preferred, sometimes it is not possible to measure directly. In these cases, indirect measures are useful.

IA is a triad of cooperation between the technology that provides assurance, the processes that leverage that technology, and the people who apply and make the technology work [7]. IA metrics should be all encompassing – the product, the process, and the people, because processes build products that people use. If we want to be assured that proper information protection is in place, we need to know what it is that we wish to protect, that we have the right product for protection, that the product was built correctly, and that the right people are using it properly.

Summary

The workshop was successful in focusing attention on the area of metrics or measures for systems that have security or assurance as a requirement. It was not successful in getting agreement on a set of measures to be used, or even finding consensus in any particular approach. Nonetheless, several themes emerged from this workshop that may be useful. These are reported below, as taken from the draft proceedings of the workshop at the time of this writing.

- There will be no successful single measure or metric that can quantify the assurance present in a system. Multiple measures will most certainly be needed, and they will need to be refreshed frequently.
- Software and systems engineering are very much related to this problem: The quality of the software delivered, the architectures and designs chosen, the tools used to build systems, the specified requirements, and more are all related to assurance.
- Penetration testing is, today, a valid measurement method. It is imperfect and to some extent non-repeatable, but nonetheless, it is used in both government and the commercial sectors. Several other testing measures are valuable: They include level of effort, numbers of vulnerabilities found (or not

found), and number of penetrations.
- There are differences between the government and the commercial sectors. One is policy driven – the other is profit driven. Defense in depth and breadth is important. Knowing how to measure this defense is also important and a valid research area. There was no agreement on how to accomplish this measurement.
- Attempts to quantify and obtain a partial ordering of the security attributes of systems in the past have not been successful to a large degree (e.g., the Trusted Computer Systems Evaluation Criteria and the Common Criteria [1, 2]).
- Processes, procedures, tools, and people all interact to produce assurance in systems. Measures that incorporate all of these are important. We believe Bodeau has characterized this very well in Figure 1 (see page 31).

References

1. Department of Defense Standard. Department of Defense Trusted Computer System Evaluation Criteria. DOD 5200.28-STD, GPO 1986-623-963, 1985.
2. ISO Standard 15408. The Common Criteria.
3. Workshop on Information-Security-System Rating and Ranking, Williamsburg, Va. 21-23 May 2001. Draft Proceedings (unpublished). <www.acsac.org/measurement/position-papers/index.html>.
4. Schuedel, G., and B. Wood. "Adversary Work Factor as a Metric for Information Assurance." Proceedings of the New Security Paradigm Workshop, ACM/SIGSAC, Ballycotton, Ireland 18-22 Sept. 2000. <wnspw.org> (ACM order number 537001).
5. National Institute of Standards and Technology, Computer Security Division, Systems and Network Security Group. Federal Information Technology Security Assessment Framework. NIST, 2000. <http://csrc.nist.gov/organizations/guidance/framework final.pdf>.
6. Bartol, N. "IA Metrics Development and Implementation." In a position paper submitted to the workshop on Information-Security-System Rating and Ranking, Williamsburg, Va. 21-23 May 2001. Booz-Allen & Hamilton, 2001.
7. McCallam, D. "The Case Against Numerical Measures of Information Assurance." In a position paper submitted to the Workshop on Information-Security-System Rating and Ranking, Williamsburg, Va. 21-23 May 2001. Logicon, 2001.

Notes

1. This work is partially sponsored by the National Science Foundation Grants CCR-0085749 and CCR-9988524.
2. Sponsored by the MITRE Corporation and the Applied Computer Security Associates.
3. The categories outlined here are from research at Mississippi State University's Center for Computer Security Research, <www.cs.msstate.edu/~security>, in a larger effort to create a taxonomy for information assurance metrics and measures. This work can be shared by contacting the authors of this article.

About the Authors

Rayford B. Vaughn Jr., Ph.D., is professor of computer science at Mississippi State University. A retired Army Colonel, he served 26 years, including commanding the Army's largest software development organization and creating the Pentagon Single Agency Manager organization to centrally manage all Pentagon information technology support. After retiring from the Army, he was vice president of Integration Services, Electronic Data Systems Government Systems. Dr. Vaughn has more than 40 publications and actively contributes to software engineering and information security conferences and journals. He holds a doctorate degree in computer science from Kansas State University.

Department of Computer Science
P.O. Box 9637
Mississippi State University
Mississippi State, MS 39762
Phone: (662) 325-2756
Fax: (662) 325-8997
E-mail: vaughn@cs.msstate.edu

Ambareen Siraj is a graduate student in the Computer Science Department at Mississippi State University and a member of the Computer Security Research Center. She is working toward a doctorate degree under the direction of Dr. Rayford B. Vaughn Jr. Her research combines the use of artificial intelligence techniques in the creation of a network-based decision engine designed to fuse and analyze information from multiple intrusion detection sensors. She also has a strong interest in the area of measuring the trustworthiness of systems and in the use of metrics and measures to do so. She has written several papers on her work and continues to be active in that endeavor.

Department of Computer Science
P.O. Box 9637
Mississippi State University
Mississippi State, MS 39762
Phone: (662) 325-2756
Fax: (662) 325-8997
E-mail: ambareen@cs.msstate.edu

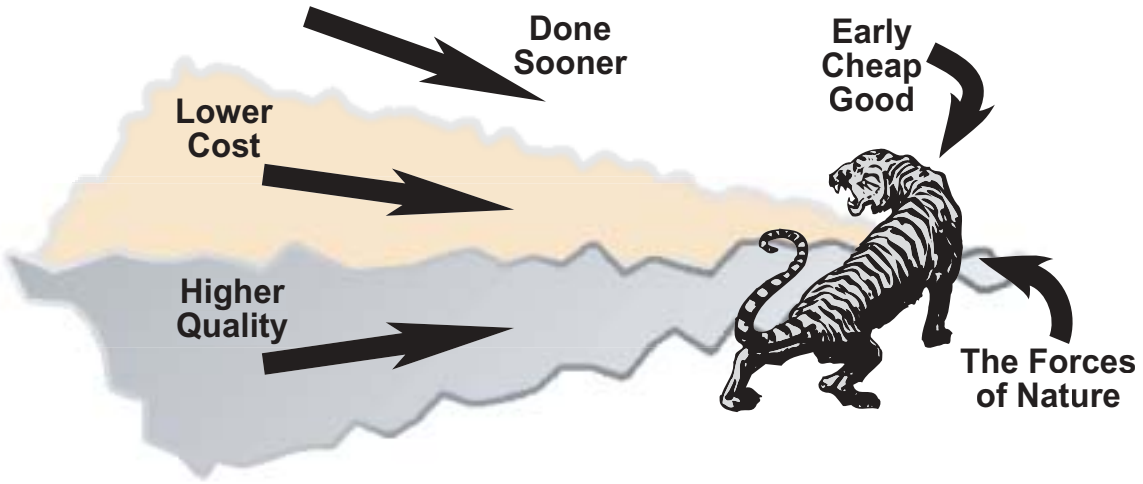
David A. Dampier, Ph.D., served over 20 years in the U.S. Army, the last 12 as a software engineer and automation officer. In that capacity, Dr. Dampier conducted research in software prototyping and software evolution at the Army Research Laboratory, and he taught software and information engineering at the National Defense University. In February 2000, he left the Army to join the computer science department at Mississippi State University where he teaches software engineering and computer science. Dr. Dampier's research interests are in formal methods for software engineering and software evolution, software process automation, and computer forensics.

Department of Computer Science
P.O. Box 9637
Mississippi State University
Mississippi State, MS 39762
Phone: (662) 325-8923
Fax: (662) 325-8997
E-mail: dampier@cs.msstate.edu

Defeating the Forces of Nature: Two Workshops on Spiral Development

Dr. Wilfred J. Hansen
Software Engineering Institute

Spiral Development (SD) and Evolutionary Acquisition (EA) are key strategies for Department of Defense acquisition. Two workshops explored these techniques and recommended a number of steps to ensure widespread enjoyment of the advantages they have shown in individual projects. These workshops and their findings are summarized in a recent article, which describes how SD and EA can help defeat the "forces of nature" that deter projects from being on time, within budget, and high quality. CrossTalk *was not able to publish the article on these workshops; however, it can be viewed at <http://www.sei.cmu.edu/cbs/spiral2000/DefeatingTheForces.html>.*



WEB SITES

Software Testing Institute

www.softwaretestinginstitute.com

The Software Testing Institute (STI) provides access to quality industry publications, research, and online services. STI offers the following professional resources: a software testing discussion forum, the STI Resource guide, the Automated Testing Handbook, the STI Buyer's Guide, and privileged access to STI's industry surveys on salaries, staffing practices, industry trends, and more.

Software Program Managers Network

www.spmn.com

The Software Program Managers Network (SPMN) is sponsored by the Deputy Under Secretary of Defense for Science and Technology, Software Intensive Systems Directorate. SPMN conveys proven industry and government software best practices to managers of large-scale Department of Defense (DoD) software-intensive acquisition programs. The SPMN 16 Critical Software Practices specifically address underlying cost and schedule drivers that have caused software intensive systems to be delivered over budget, behind schedule, and with significant performance shortfalls. They provide "hands-on" consulting, training, and direct support to DoD programs in information technology project management, requirements and risk management, and more.

Quality Assurance Institute

www.qaiusa.com

The Quality Assurance Institute (QAI) is an international organization dedicated to partnering with the enterprise-wide information quality profession in search of effective methods for detection-software quality control and prevention-software quality assurance. QAI provides consulting, education services, and assessments.

Software Technology Support Center

www.stsc.hill.af.mil

The Software Technology Support Center is an Air Force organization established to help other U.S. government organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

Agile Modeling

www.agilemodeling.com

The Agile Modeling site was established to develop and promote the agile modeling (AM) methodology. It contains sections that explain AM: its goals, scope, and an overview of the values, principles, and practices of AM.

LETTERS TO THE EDITOR

Dear CrossTalk Editor:

Don Reifer states in his article [CrossTalk, Mar. 2002] "Let the Numbers Do the Talking" that he is getting tired of being misquoted. So am I. In my CrossTalk article,[1] I did not state or imply that an uncalibrated software cost model would outperform a calibrated model. All estimating models must be calibrated and validated. The point I raised is that when a user recalibrates an instantiation of a cost model, the instantiation must be revalidated as a new model. The models no longer have the same characteristics. For example, REVIC (a recalibration of COCOMO) and COCOMO are not the same estimating model. Because of the differences between the original and recalibrated models, one cannot compare estimates made by COCOMO and REVIC.

Whether a cost model is calibrated or not, when they are put in the hands of untrained, inexperienced estimators, you still get really poor estimates.

Dr. Randall W. Jensen
President, Software Engineering, Inc.
E-mail: seisage@aol.com

1. Jensen, R. "Software Estimating Model Calibration." CrossTalk July 2001: 13-18.

Dear CrossTalk Editor:

Since my article "Let the Numbers Do the Talking" was published in the March 2002 CrossTalk, two questions relative to Table 2 and Table 3 keep popping up in e-mails sent to me. I hope the following clarifications will resolve the issues for everyone interested.

Table 2 – The cost per staff month of $12,000 reflects burdened labor cost exclusive of G&A and profit for typical inexperienced labor mixes across both military and commercial domains. This is the internal cost of a person. It is not the price charged when the services of a person are sold to a third party (another division, government customer, etc.).

The price per person-month varies greatly by industry because of skill mix, experience, and business practices. For example, the average price per person-month for a regular software engineer within the aerospace industry ranges from $21,000 to $25,000/person-month. This price reflects a more experienced staff mix than what is normal in the commercial world and additional markups to reflect the high costs of doing business with the government.

In contrast, the internal price that organizations within the telecommunications industry charge other parts of the same firm for services can be as low as a base cost of $12,000 to $15,000/person-month. When the labor mix is young, profit doesn't enter into the calculation, and only a very small markup is charged for doing business with a sister organization.

Table 3 – SRR in this table refers to Software Requirements Review, not Systems Requirements Review.

Donald J. Reifer
Reifer Consultants, Inc.
E-mail: dreifer@ieee.org

First-of-Its-Kind Software Development Certification Exam Offered

After a rigorous three-year development process, the Institute of Electrical and Electronics Engineers (IEEE) Computer Society will offer its new Certified Software Development Professional (CSDP) examination twice in 2002: Apr. 15-June 30 and Oct. 5-26. The new CSDP credential is intended for mid-level professionals. The 3.5-hour examination covers topics such as software design, software requirements, and software testing. Candidates are required to have a baccalaureate or equivalent university degree and a minimum of 9,000 hours of experience in at least six of 11 knowledge areas. The CSDP is designed to elevate educational standards and recognize those who demonstrate knowledge essential to the practice of software engineering.

The certification examination is the first in a series to be offered under the society's "Doing Software Right" initiative. The IEEE Computer Society developed the CSDP exam in conjunction with the Chauncey Group International, a leading certification test consultant and subsidiary of the Educational Testing Service. The exam will be offered at more than 300 testing centers in the United States and Canada, as well as in select cities in Brazil, China, Hungary, India, Ireland, Japan, and Russia.

The Computer Society has also published a new two-volume resource guide for the CSDP program, an updated and expanded version of the best-selling CS Press tutorial, Software Engineering. Further information about the CSDP examination and the CSDP preparation program is found at <http://computer.org/certification>.

Week of the Geek

The other day, I was saving some data to my favorite backup media – which happens to be a 128 Meg SmartMedia card. The card is small and has the capacity of 88.8888 … (Oh heck – let's round it off to 90) floppies. I carry the card in my sunglass case. A friend saw me pulling the card out of my glass case and called me a "geek."

Me? A geek? Probably. I'm not ashamed of it. In fact, I think I'm a bit proud of the title. But do I look like a geek? In fact, how do you tell who the geeks are? There used to be certain indicators that you were a geek. The best sign used to be black plastic eyeglass frames (with white tape holding them together at the nosepiece). Now, thanks to laser eye surgery, geeks don't need to wear them anymore. Also – thanks to retro fashions – lots of people who are not geeks are wearing black plastic frames, which happen to be in fashion. (Sure, now that I don't wear them anymore!)

Another sign used to be a plastic pocket protector full of pens and pencils. Nowadays, I own a single all-in-one writing instrument that has a palm stylus, black pen, red pen, and pencil. No pocket protector needed. And in the very old days, a dangling slide rule at the belt was also a prerequisite of geekhood. Nowadays, slide rules dangle in the Smithsonian.

What we need today are contemporary indicators of being a geek. After some thought, I submit the following list as reasonable indicators of geekiness:

- You have 50 people in your online address book, but only three have real addresses.
- You send more e-cards than real ones.
- Some of your best friends are people you have never actually met in person.
- When in a bookstore, you pick up an "X for Dummies" book, you read a bit, laugh, and say, "Nobody could really be THAT dumb."
- You automatically add a "com" after a period when typing.
- When you introduce yourself, you include your e-mail address.
- You attend a conference and automatically look for a seat near an outlet, so you can plug in your computer.
- When you travel, you already know where outlets are located in the airline gate areas.
- A good hotel is defined as one where you can get a 52K bps connection (bonus points if you know which hotels in advance).
- You have e-mail addresses for different facets of your personality. For example, david.cook@hill.af.mil for work, and brainy_stud@someISP.com for home. (Only the first one is actually mine.)
- You frequently wish life had an undo or back key.

- When your significant other says you need to communicate better, you think that means you can get DSL.
- By looking at the control panel applications, you unconsciously determine whether the computer is running 95, 98, ME, 2000, or XP.
- You go to conferences where well-dressed guys wear suits or sports coats with very short high-water pants and running or tennis shoes. Extra points for white socks. Double points for white socks with black or navy pants.
- You have gotten up in the middle of the night to check the status of either a big download or a disk defrag.
- You wonder whether you can daisy chain USB port replicators to give you more than four USB connections per computer port. Extra points if you really need more than four USB connections per computer port.

Recently, I was at the Software Engineering Process Group conference in Phoenix. During a fine southwestern pork lunch, I asked friends at my table to help me focus on signs that would ping a geek meter. The following signs were mentioned (thanks to John, Lisa, Les, Jim, Janna, Nicole, Jeff, Marsha, and Kate):

- You know all the "Star Trek" plots (with emphases on classic Trek, not TNG). You utter such phrases as, "He's dead, Jim" and "I'm a doctor, not a (fill in the blank)," and your friends laugh.
- You live in the West or Southwest, and the people at Fry's electronic superstore know you by name. (If you live elsewhere, CompUSA will substitute.)
- You know the words to most Monty Python songs, and you can quote sections of "Monty Python and the Search for the Holy Grail" from memory.
- You can also recall quotes from *The Hitchhikers Guide to the Galaxy*.
- Your watch has more dials, buttons, and functions than a Swiss army knife. Double extra bonus points for having a wristwatch that automatically synchronizes itself with the National Bureau of Standards via short wave. Triple bonus points if you're thinking, "Cool! I want one, too!"

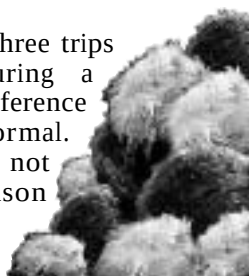
Do you know other signs of being a geek? Well, the theme of the December 2002 issue of CrossTalk is "The Year of the Scientist and Engineer." If you will e-mail me your geek stories and indicators, I will compile them and have an appropriate BackTalk for the December issue. Send them to me at david.cook@hill.af.mil. We'll have a "Top 10" geek indicators list.

To conclude, if December 2002 marks the "Year of the Scientist and Engineer," I hereby proclaim the Software Technology Conference 2002 to be the "Week of the Geek." Hope I see you there.

By the way, one sign of being a geek might be that you wrote your BackTalk column on a palm computer while attending a computer conference. Mind you, I'll just call it good time management!

—David A. Cook, Geek
Software Technology Support Center

P.S. By the way, three trips to Fry's during a four-day conference is not abnormal. And that was not the only reason I decided to attend.



Sponsored by the Computer Resources Support Improvement Program (CRSIP)

Published by the Software Technology Support Center (STSC)

CrossTalk / TISE
7278 4th Street
Bldg. 100
Hill AFB, UT 84056-5205

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737