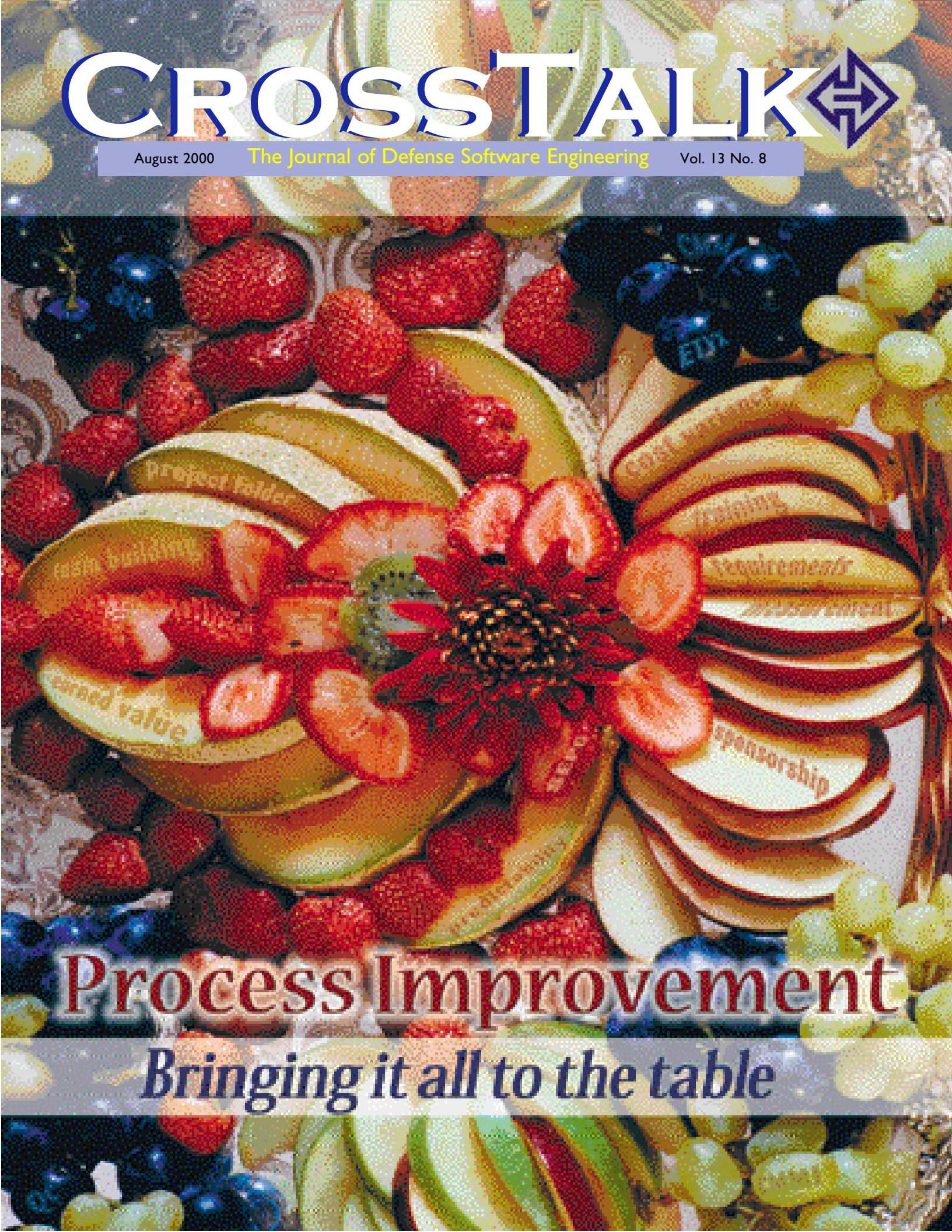

CROSSTALK
August 2000
The Journal of Defense Software Engineering
Vol. 13 No. 8
Process Improvement
Bringing it all to the table

Process Improvement

On the Cover: Larry W. Smith brings it all to the table with his latest cover—he is a graphic artist as well as an engineer for the STSC.

Software Engineering Technology

Software Engineering Education

Open Forum

Departments

CrossTalk

SPONSOR	*H. Bruce Allgood*
PUBLISHER	*Reuel S. Alder*
ASSOCIATE PUBLISHER	*Lynn Silver*
MANAGING EDITOR	*Kathy Gurchiek*
ASSOCIATE EDITOR/LAYOUT	*Matthew Welker*
ASSOCIATE EDITOR/FEATURES	*Heather Winward*
GRAPHIC DESIGNER	*Abby Hall*
VOICE	801-775-5555
FAX	801-777-8069
E-MAIL	crosstalk.staff@hill.af.mil
	www.stsc.hill.af.mil/
	Crosstalk/crostalk.html
	www.crsip.hill.af.mil

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may use the form on page 31.

Ogden ALC/TISE
7278 Fourth Street
Hill AFB, Utah 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 *Guidelines for CROSSTALK Authors*, available upon request. We do not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

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

Trademarks and Endorsements: 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.

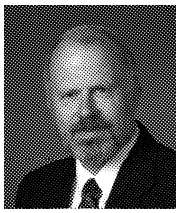
Coming Events: We often list conferences, seminars, symposiums, etc., that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CROSSTALK Editorial Department.

STSC Online Services: at 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 Makes Software Process Improvement Happen?



The Department of Defense spent an estimated $35.7 billion on software in 1995 and it will spend $20 billion annually on embedded software alone beginning in 2002 [1]. When software process improvement is applied to all software efforts, the savings could be $5 billion per year or better. This is based on government and industry data collected over the past 10 years.

One example is the Software Engineering Division at Hill Air Force Base, Utah, which announced its Level 5 Capability Maturity Model (CMM®) achievement and reported a 35 percent cost savings in its software maintenance activities [2]. Industry and government sources have reported cost savings from 9 percent to 67 percent, reductions in software errors from 10 percent to 94 percent, and schedule compressions from 15 percent to 23 percent [3].

In addition to the primary benefits to process improvement there are significant secondary benefits. These include improved employee morale, less overtime, fewer crises, less absenteeism and less attrition. Less attrition can be a significant savings to software organizations that experience a 16 percent industry-wide turnover rate, according to Boeing Information Systems data. Organizations with well-developed processes show 3 percent attrition.

The business case for software process improvement is well established but how does it happen? It must begin with top management sponsoring the effort. This means a commitment of management time and corporate resources. A blessing by top management with a launch-and-forget mentality will fail. Process improvement requires the time and attention of senior leaders. They must attend the planning sessions, the report and review sessions, as well as the decision sessions.

They must approve the selection of the organization's best performers to conduct the improvement effort. The quality of the individuals selected to staff technical working groups is a demonstration of management's commitment to succeed by engaging those most likely to succeed. George Jackelen's article on page nine addresses some of the issues about getting started. Participation was slow initially. A common understanding of the project, the goals, and definitions were needed. It takes time to focus an organization on process improvement. There is a period of buy-in where employees test management's resolve. The question weighing on their minds is how serious is management. Is this just another quick fix, feel good activity that will pass is a few days or months?

In addition to sponsorship, the culture and capabilities of the organization need to be considered. Beginning on page four, Tim Kasse and Dr. Patricia McQuaid discuss many issues affecting process improvement. Does the organization have a history of resistance to change? Do they have qualified human resources? Process improvement is not a substitute for engineering capability. A Level 1 organization can at least deliver software; they are just inefficient at doing it. Teaching those who are software illiterate the virtues of CMM is entertaining but not productive.

There are cultural issues to change. Marriages between two cultures often fail. Neither partner has a basic understanding of the other's needs. Organizations develop a cultural pattern. They have their unwritten rules of order. When these rules are violated, change becomes difficult. When they are understood and applied, change flows more quickly and efficiently.

Progress toward a goal is best achieved when the goal is known and improvements are measured. A CMM-based Appraisal for Internal Process Improvement is an important milestone. A team of people composed from inside and outside the organization conducts this assessment. What an organization learns from this assessment will focus their efforts on what needs to be done.

Reuel S. Alder
Publisher

1. Sanders, Dr. Patricia. Improving Software Engineering Practice, CROSSTALK, January 1999, Vol. 12, No. 1.
2. Oldham, Leon, et al. Benefits Realized from Climbing the CMM Ladder, CROSSTALK, May 1999, Vol. 12, No. 5.
3. Griffin, Scott. Boeing's Continuous Improvement Journey, SEPG 2000.

Factors Affecting Process Improvement Initiatives

By Tim Kasse
Kasse Initiatives LLC

and Patricia A. McQuaid,
California Polytechnic State University

When starting a process improvement initiative, it is imperative to determine the appropriate tasking and the scope of the process improvement program. It is tempting for an organization to try to take on too much too fast, especially if it feels that it must catch up to its competition. While it is natural to want to initiate a program quickly, it is important for an organization trying to get a process improvement initiative started to be as realistic as possible in these beginning stages. This paper will identify the critical factors to consider that have helped the authors' clients.

This paper is based on the authors' previously published, "Entry Strategies Into the Process Improvement Initiative," *Software Process Improvement and Practice Journal,* Vol. 4, Issue 2, pp.73-88, June 1998.

The Critical Factors Identified

When an organization is trying to establish its organizational process, there are many factors that must be taken into consideration. We have identified the following factors that we believe are critical for organizations that have started their process improvement programs in the past 10 years, and will discuss them throughout the remainder of this paper.

- History of previous process improvement programs or quality improvement programs.
- Financial resources to fund the process improvement initiative.
- Human resources that can be dedicated to process improvement.
- Software engineering capability of the developers.
- Technology support available.
- Contractual obligations.
- Scope.
- Customs and culture of the organization.
- Standards (industry, corporate, organizational, project, customer).
- Understanding and support from all levels of management and practitioners.
- Corporate political pressure.
- Business objectives.
- Vision.

History

Knowing the history of previous process improvement or quality improvement programs can help the process improvement initiative an organization is about to undertake. After one or two intitiatives associated with quality management or process improvement have failed, an organization becomes disinterested and cynical. Existing process improvement attempts, perhaps scattered throughout the organization, should be examined to determine if they could be used as a good starting base for this new organizational process improvement effort. Looking at what went right and what went wrong with past process and quality improvement attempts can be very useful. If an approach or part of an approach was successful, perhaps following that approach will be successful again. If an approach was not successful, then understanding what caused the failure could help avoid those problems and negative feelings the next time. Understanding history can help ensure future success.

Financial Resources

Most organizations that get involved with process improvement are in business to make a profit. Process improvement is not free; it takes money, computer resources, human resources, tools and techniques, training, and consulting support. It is important to determine how much process improvement an organization can afford. Regardless of what standard or model is followed or what external pressure may be exerted on the organization, there is a real probability that it can only afford so much process improvement. Even if the managers and practitioners of the organization wanted to improve in multiple areas at one time, they may be financially restricted to focus on a few key areas and add others as they begin to show a measurable return on investment.

Human Resources

When we speak of human resources, we must also state that we are talking about qualified human resources. A process improvement initiative that will be successful must be able to assign or find qualified process improvement agents. Some of the best candidates for the support of process improvement may also be the key players in development or project management. Senior, especially middle, management normally is not happy to move these people out of their perceived strategic positions and into process improvement. Finding qualified process improvement agents to join an organization, to come up to speed on its products and services, and to adapt to its culture is also a difficult task. Process improvement requires strong process improvement or change agents who can coach and mentor others, allowing them to become comfortable with the changes and even more productive. The process improvement agents or Software Engineering Process Group members need to understand senior management's strategic direction, the organizational structure, software support activities such as Software Quality Assurance and Software Configuration Management, modern software engineering techniques and methods, and basic project management practices in order to increase the likelihood of success. They also must be able to work within the organizational culture, know how to manage technology change, be able to apply team-building concepts, and utilize collaborative consulting skills.

Understanding Software Engineering Principles

It is worth taking the time to understand the organization's comprehension of basic software engineering principles and deciding how much training, coaching, and mentoring will be necessary to support the process improvement initiative. It might be wise to provide some basic software engineering training and follow up with coaching and action planning as the organizational members realize what it is that they need to do to improve.

The Capability Maturity Model (CMM®) and the associated process improvement aids that the Software Engineering Institute (SEI) supports assume that the organization has a sound understanding of software engineering principles but does not apply them very well. CMM Level 2 Key Process Areas focus on project management practices to remind managers and developers that they need to adhere to these project management practices to properly manage and control their projects. The CMM is not a Software Engineering Handbook. It does not describe the underlying software engineering principles, but assumes that they are understood. Asking for an assessment and assuming the organization can accept the assessment results, develop an action plan, and start to implement it with positive measurable results can be very risky! It is useful to determine the level of understanding of software engineering at all levels of management, as well as to determine the software developers' level of experience and understanding of software engineering principles.

Technology Support

Another critical element is to determine the organization's attitude toward technology. It is important to ascertain whether technology is sought after as the "silver bullet" or if it used to support the process and make the developers more productive. Where does the technology support come from now? Does an in-house group provide it or is it provided by an outside vendor with a long-term contract that is difficult to break or get around? What budget is available for technology?

If management views technology as a quick answer for the organization to get on with its important work, or sees technology as a way to increase the developers' productivity by 15 percent to 20 percent without any other support, then it is important to help management understand process improvement concepts. However, if management's attitude is that appropriate technology should be used to support the process improvement initiative and that both process and technology are needed to allow workers to be as creative and productive as they can be, then an emphasis on technology is desired.

Technology is necessary to support the managers and developers working on today's complex systems. A quick glance at the CMM might give one the impression that technology is not thought about until Level 5, but of course that is not true. Technology is required to support the managers and developers at every level; however, the technology must complement the process, not drive it. Any entry strategy into software process improvement must take past, present, and future technology into consideration.

Contractual Obligations

The talk about process improvement seems to indicate that an organization needs merely to decide that it will undertake a process improvement initiative and all other factors are incidental. However, some of the organization's major projects may have long-term contracts that can be viewed as blocking factors to process improvement. Some or many of the process improvement ideas may not be allowed to be implemented on those projects without first obtaining permission from the customer. This implies that either an understanding or an agreement must be made with the customer for that existing contract or the organization may be restricted in what it can realistically implement on such a project.

Scope

The term *scope* is most appropriately used when preparing for an assessment or evaluation (audit). You must decide whether the entire organization is to be examined or just a department or even a project or two. It is entirely appropriate to think about the scope when preparing for a process improvement initiative. Each process improvement initiative must have a clearly defined scope. Is it the intent to institutionalize the management practices of Level 2 throughout the organization? From a political or strategic point of view, is it more appropriate to focus on a product line? What departments should be included in the process improvement initiative? Are there specific projects that should spearhead the process improvement effort and then gradually expand successful concepts to the rest of the organization? The scope of the process improvement initiative will also determine how many resources are needed to support this effort. Resources include people, money, equipment, tools, methods and techniques, training, and potentially outside support from consultants.

Customs and Culture

Culture may seem obvious to us, but it is subtle and can have an enormous influence on the way that we think and go about our daily lives. Organizational culture not only affects the development and maintenance of software but also everyday morale of the work force. Culture may come from the region of the country or world that we are living and working in. The culture of the European countries of France, Spain, and Italy is very different from that of Germany, the Netherlands, and Scandinavia. The culture of the southwest region of the United States is vastly different from that of the East Coast. Some cultures conduct meetings only to tell the participants what decisions have been made, and others are so team oriented that an answer given to an outsider is almost always the result of team thinking. Some cultures are open and accept challenges and innovations, whereas other cultures are closed and decisions must come from the top.

Much of the process improvement thinking today seems to support the idea that a strong process focus should transcend the personalities of the top management of an organization. In the authors' experience it is more often that the senior management team's attitude toward quality and process improvement has a strong influence in determining what the organization's culture will be. The organizational process improvement initiative must

take into consideration the organization's culture, if this initiative is to have any chance of success.

Standards

The various standards established for software development around the world have their own special effects on a process improvement initiative. Department of Defense (DoD) standards such as MIL-STD-498 and previously MIL-STD-2167 and MIL-STD-2168 directed ways of approaching problem solving, prescribed formats of documentation, and dictated software life-cycle approaches that influenced many software support activities such as configuration management and software quality assurance [1], [2], [3]. International Organization for Standardization (ISO) standards such as ISO 9001, ISO 9004, and ISO 9000-3 (TickIT) are a set of commonly talked about international standards. They were intended to allow countries to trade with one another and be able to expect a certain level of quality [4], [5], [6], [7]. Instead, the pressure to be ISO certified pushed organizations to develop pages of documented processes that were rarely known or used throughout the organization. Institute of Electrical and Electronics Engineers (IEEE) standards provide templates and guidelines for organizations to adapt to support their software development efforts [8]. The Capability Maturity Model, developed at the SEI in Pittsburgh, has become a de facto standard in the world [9], [10], [11]. Many companies, however, try to apply the CMM as an absolute model rather than the roadmap or guide that it is.

Your organization's requirements to adhere to a standard and/or its reaction to standards can have a profound influence on your process improvement initiative.

Understanding and Support from All Levels of Management and Practitioners

From Dr. Edward Deming's ideas on Total Quality Management to the texts that are found today on quality management and process improvement, it is clear that management understanding and cascading management support is critical for the success of any process improvement initiative [12]. Without senior management sponsorship, process improvement change is slow at best. However, it is most frequently the middle managers who become the blocking factors in any quality management or process improvement effort. Middle managers today are under tremendous pressure from multiple sources: they feel pressure from the senior management team to produce products better, faster, and cheaper; and they feel pressure from the developers and first-line managers to provide the latest in computers, languages, methods, and techniques. Many of the middle managers are not aware of the processes that their developers are following. As a result, they often resist process improvement initiatives due to this lack of awareness and to the pressures just noted. Therefore, any process improvement initiative must take these factors into consideration. The senior management team must encourage, train, and support middle managers in process improvement principles so they know what is in it for them and, in turn, provide the necessary support from their level.

Although the CMM puts tremendous emphasis on management practices, a successful process improvement initiative must also have the practitioners' support. Groups, managers, and practitioners must be trained, mentored, and coached.

Corporate Political Pressure

While there may be companies that are unaware of the CMM or do not care to use it as a guide for their process improvement initiative, the political pressure to be CMM Level 2 or CMM Level 3 by a certain time is at epidemic proportions throughout the world. Many large international companies have edicts from corporate offices indicating that each business unit must achieve SEI CMM Level 2 or SEI CMM Level 3 in 12 months or 18 months. The idea of process improvement frequently gets lost. With at least a few of the large clients that the authors are aware of, a vice president is offered an incentive of $10,000 to $20,000 if his/her business unit achieves the CMM level number. Some of these vice presidents have issued orders to their process improvement managers to do whatever it takes to get the certification of the needed level—process improvement is secondary.

One European client felt this corporate pressure so badly that the client abandoned its systematic approach of process improvement and resorted to developing processes in isolation from those who would use them, providing a two-hour overview training, and declaring the practice to be institutionalized. The resentment to the process improvement effort and to the CMM was the highest these authors have seen in 10 years. The individual managers and practitioners could not see what benefit they were getting if the only goal was to achieve a level. Eventually the management team took a stand against the corporate directive and backed a process improvement initiative that would support the business and eventually result in a higher maturity level rating. Political pressure must be taken seriously when starting a process improvement initiative.

Business Objectives

For a process improvement initiative to be successful, it is imperative that it is tied to the organization's business objectives. Typical business objectives have included reducing time to market, improving delivery-time accuracy, increasing the quality of products, and increasing market share. It is important to determine the organization's highest priorities; the consequences to business resulting from weak or ineffective processes, and the action taken to correct the cause. You need to identify how the process improvement initiative is seen to support the organization's business objectives, and how the process improvement initiative will tie in to the organization's overall focus on Quality Management. Only then can the management practices at CMM Level 2 and the technical practices at CMM Level 3 take on real meaning. For example, if a business objective is, "Find and fix each problem once," it can be shown that applying the management practices of Software Configuration Management will support this business objective.

By getting management to define the business objectives, the process improvement initiative can be adapted to support those business objectives. By using the CMM as a

roadmap, an organization can accomplish process maturity and use this maturity to support the business objectives; otherwise maturity levels are often useless. Be careful not to lose sight of the business goals, thinking that you are going through the process only to attain a certain level regardless as to whether the process works for your organization.

Vision

Understanding senior management's vision is a critical step in establishing an organization's process improvement initiative, one whose value cannot be emphasized enough. In the past three years, assessments have revealed that a lack of understanding of the senior management's vision has caused measurable lack of motivation and productivity.

Where senior management thinks the organization will be in the next year, and in the next two to five years, must be identified. Competitors and strategic alliance partners need to be recognized. The technology changes that are expected and/or will be required to support the vision should be addressed. Determine the necessary organizational structure to support this vision, as well as what the organizational culture must be. Only then can you determine how a Process Improvement Initiative will support this vision.

Conclusions

Getting involved with process improvement is essential for companies that develop software today. However, deciding on how to get started, how many resources to dedicate to this effort, understanding how it supports the organization's business objectives, and many other factors, is not an easy task.

The factors that may affect a process improvement initiative must be determined and used to guide the organization into choosing the right entry strategy for it. One size does not fit all and an assessment may not even be the right place to start! The first step toward a successful improvement program is to choose the appropriate process improvement entry strategy.◆

References

1. MIL-STD-498, Software Development and Documentation, May 12, 1994.
2. MIL-STD-2167A, Defense System Software Development, Feb. 29, 1988.
3. MIL-STD-2168, Defense System Software Quality Program, April 29, 1988.
4. ISO 9001:1994 Quality Systems—Model for Quality Assurance in Design/Development, Production, Installation, and Servicing.
5. ISO 9000-3, Quality Management and Quality Assurance Standards—Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply, and Maintenance of Software.
6. ISO 9004-1:1994, Quality Management and Quality System Elements—Part 1: Guidelines.
7. ISO 9004-4:1993, Quality Management and Quality System Elements—Part 4: Guidelines for Quality Improvement.
8. IEEE, IEEE Standards Collection on Software Engineering, 1994 Edition, IEEE Inc.
9. Paulk, M.C., Curtis, B., Chrissis, M.B., *Capability Maturity Model for Software, Version 1.0,* Software Engineering Institute, CMU/SEI-91-TR-24, August 1991.
10. Paulk, M.C., Curtis, B. Chrissis, M.B, and Weber, C.V., *Capability Maturity Model for Software, Version 1.1,* Software Engineering Institute, CMU/SEI-93-TR-24, February 1993. [This report combined with 24 is referred to as the CMM (version 1.1)].
11. Paulk, M.C., Weber, C.V., Garcia, S.M., Chrissis, M.B., Bush, M., "Key Practices of the Capability Maturity Model, Version 1.1", Software Engineering Institute, CMU/SEI-93-TR-25, February 1993. [This report combined with 23 is referred to as the CMM (version 1.1)].
12. Deming, W.E., *Out of the Crisis,* MIT Press, 1982.

About the Authors

Tim Kasse serves as the manager and principal consultant of Kasse Initiatives LLC. Previously he served as the Chief Executive Officer and Principal Consultant for the Institute for Software Process Improvement that he co-founded with Jeff Perdue in 1991. His focus is on innovative solutions for process improvement of business, systems, software, people, and lifestyles. Kasse spent four years at the Software Engineering Institute and was a major contributor to the development of the Capability Maturity Model, which provides the framework for the SEI's assessments and evaluations.

United States
Kasse Initiatives, LLC.
30 West Sheffield Ave
Gilbert, Ariz. 85233
Voice: 602-855-1101
Fax: 602-855-1119
E-mail: kassetc@aol.com

Europe
Snijderstraat 23
5345 PC Oss,
The Netherlands
Voice: 31-412-692-444
Fax: 31-412-692-787
www.kasseinitiatives.com

Dr. Patricia McQuaid is an associate professor of Management Information Systems at California Polytechnic State University. She has taught a wide range of courses in both the Colleges of Business and Engineering, has industry experience, and is a certified Information Systems Auditor (CISA). She is serving as the North, South, and Central American Chair for the Second World Congress for Software Quality, slated in Japan in September. Her research interests include software process improvement, software quality, and software testing.

Dr. Patricia McQuaid
California Polytechnic State University—MIS Area
College of Business, San Luis Obispo, Calif. 93407
Voice: 805-756-5381
Fax: 805-756-1473
E-mail: pmcquaid@calpoly.edu

✎Letters to the Editor✎

I have used back issues of CROSSTALK often in my software process improvement work both at Xerox and Hughes. It is really helpful!

—Delores J Harralson, Hughes Space and Communications

I'm having withdrawal . . . not having CROSSTALK to read since I left Puget Sound Naval Shipyard! The hard copy will be perfect to read on the Metro on my way to and from work!

—Cathy Ricketts, NAVSEA

Process Improvement Web Sites

www.sei.cmu.edu/publications/documents/96.reports/96.ar.biblio.softproc.impr.html
This is a bibliography for software process improvement written by Mark C. Paulk. The bibliography lists studies and experience reports on the effects of software process improvement.

www.sei.cmu.edu/publications/documents/96.reports/96.hb.001.html
"IDEAL: A User's Guide for Software Process Improvement (SPI)" by Robert McFeeley describes "a SPI program model, IDEAL, which can be used to guide development of a long-range, integrated plan for initiating and managing a SPI program. The purpose of this document is to provide process improvement managers with a generic description of SPI."

www.espi.co.uk
This is the ESPI Foundation's site, which it says is "dedicated to continuous software process improvement and increased business performance." There are links to the European SEPG™ technical workshop, and to SPIshare.

www.ispi.co.uk/ispi71.htm
This is the site for the Institute for Software Process Improvement, with links to SEI CMM, SPI, and change movement. Web pages are dedicated to such things as Process Impact—Software Process Improvement Consulting and Education; and Software Process Improvement and Change Management: Keys to Success.

www.sei.cmu.edu/collaborating/spins/spins.html
The Software Process Improvement Network is made up of individuals who want to improve software engineering practice, and those individuals are organized into regional groups called SPINS. This site links to a directory of U.S. and international SPINs, announcements, and ways to start or join a SPIN.

www.hio.hen.nl/~zielman/compsci/seng/spi.html
This site offers links to SPI, such as the Software Engineering Institute home page, a critique of SEI's CMM, the SPIDER project, and the Managing Process Improvement Course.

http://web.mit.edu/lfm/www/working_papers/1998_abstracts/cjohnson_abstract_1998.html
This site provides readers with an abstract written for a thesis on the "Dramatic Improvement of a Mature Process: Proactive Process Improvement." There also is a link to the Class of 1998 theses list, working papers, and a home page.

www.computer.org/software/so1997/s5075abs.htm
Readers will be linked to a site on "How Software Process Improvement Helped Motorola." The authors of this paper, copyrighted in 1997 for the Institute of Electrical and Electronics Engineers Inc., offer metrics and data that show the results of Motorola's usage of the Capability Maturity Model.

Other IEEE-copyrighted articles may be found at

www.computer.org/software/so1998/s1064abs.htm

www.computer.org/software/so1999/s3037abs.htm

IEEE Computer Society membership may be required to access.

www.sqi.gu.edu.au/spice/title.html
Welcome to the Web site by Software Process Improvement and Capability dEtermination (SPICE). There are links to an abstract on "Software Process Improvement in Web Time," an introduction to software process improvement, strategic business planning and process improvement, and the Engineering Process Improvement Center of Lockheed Martin Corp.

www.marotz.com/journal/dec98/links.htm
"Useful Process Improvement and Project Management Links" takes readers to various newsletters as well as short courses curricula.

Software Best Practice Development–An Experience

By George Jackelen
EWA Inc.

A Level 3 process capability requirement includes having documented software processes (best practices)[1]. This paper addresses an experience of initiating a process development effort.

Over the last several months, our prime contractor has been working on an initiative to develop documented best practices on how we do software Independent Verification and Validation (IV&V). The scope was narrowed to our current work, e.g., document development, review of customer supplied documents, test witnessing.

This paper provides some information about the work and decision making we do to get started and to develop a first set of project-level best practices. For instance, do we use a top-down approach (i.e., start with policies and work down to instructions) or did we use a bottom-up approach (i.e., start with instructions and work up to policies)?

The following describes some of the issues raised (not in chronological or priority order) and the eventual decisions, which could change. As with all decision-making efforts, the results do not apply to every environment. In some situations people may challenge a decision, but consensus and management direction are often more important than the actual decision.

This paper does not mean the process work we started has ended. It has just begun.

Getting Started

The IV&V Project Manager (PM) created a Process Development Group (PDG) consisting of eight people: a PM-appointed lead, a member from each organization, a person with experience in procedure development and implementation, and a person from the ISO 9000 development group. The PM attended almost every meeting to facilitate the process. Once the PDG created a Process Instruction (PI, described later), the PI would be distributed among the organizational leads and then to all the project members for comment and acceptance. The following are some interesting PDG characteristics:

- During the first three months, only two members participated in more than 90 percent of the e-mail discussions on how the Process Development Group should operate, the format and style of the PIs, and comments on the first PI. Attempts by the two members to insert controversial statements did not invoke discussions.
- During the weekly meetings, four people provided more than 90 percent of the discussion. When open-ended questions were asked to solicit opinions, the other four rarely responded. However, when the PDG Lead met with individuals one-on-one and asked them questions, they would respond, sometimes disagreeing with a meeting's decision.

Many organizations use the following hierarchy to describe their best practices: policies, procedures, work instructions, and checklists to insure the procedures and work instructions are implemented. Instead of developing one or more policies for our group, we agreed to use the Annual Business Plan (ABP), (an agreement between IV&V and our customer). This also gave us flexibility since the ABP is revised annually to provide direction for IV&V during the upcoming 12 months.

Another get-started issue was determining the level of detail needed to provide direction to the project IV&V software engineers. For instance, should we develop procedures and work instructions? Part of the discussion resulted in the need to define the difference between procedures and work instructions. Some believe procedures are developed to provide process instructions stating what needs to be done to satisfy group interfaces; work instructions provide directions to individuals on how to implement their part of the procedures. Others believe there is no difference. Due to the engineers' talent, the decision was made to only develop PIs—a compromise between procedures and work instructions—and to provide just enough instructions for the engineers to know what to do, allowing them to provide the details. This provides flexibility and reduces the need for too many procedures and related documents.

Glossary and Terminology

Some people wanted each PI to contain the definitions it used. The consensus was to have a Process Dictionary for all terms and abbreviations. PIs would be allowed to modify a term in this list to satisfy their particular needs. PIs could also include terms not provided in a Process Dictionary. Definitions of terms normally found in a dictionary would not be part of a glossary. Draft PIs would include terms, definitions, and abbreviations not found in the Process Dictionary; before a PI was finalized, there would be a decision to add new terms to the Process Dictionary or leave them in the PI.

Terminology was a related issue. For instance, some people did not know the difference between a report (a document on results of an analysis or assessment) and a plan (a document describing methodologies, schedules, resources, etc., to be implemented). Another example was the belief that a walkthrough was the same as a review. Also, people did not realize they had different definitions of "peer group." These issues reinforced the need for a common documented set of terms and definitions.

Configuration Control

Discussion included the need to control the PIs, e.g., distribution of the latest version. The agreement was to have each PI maintained (responsible for creation, modification, and termination) by a maintainer/process owner. The process owner (the preferred name) would ensure each PI version was coordinated with the PM and all organizations involved with the PI. A new version would be published when there was a major change to a PI or the PI's process owner changed (e.g., a change in personnel). This would ensure that the new process owner was aware

of the PI and provided a chance for the new process owner to coordinate any changes he wanted to implement).

PI Format

The following summarizes discussion about the PI format:

- Title of the process owner and date of implementation for this PI version would be at the top of page one.
- The header would contain the PI's title.
- Footer:
  - –A PI identification number would include a version identifier. We agreed to use the document identification number to control the draft PI version. The PI identification number format would be "X.Y," where X is the approved version number and Y is the sequential draft number.
  - –Page number to consist of "Page X of Y," where X = current page number and Y = total page count.
- PI date. The discussion was what this date represents. The decision was that this would be the date of first implementation of this PI, immaterial of how many revisions occur. For draft PIs this would represent the date of the draft.
- *Purpose* would be the first PI section, describing (normally in one to three sentences) the purpose of the PI.
- A *scope* section defines the limitations/boundaries of the PI.
- A *glossary* section would only define terms not provided in the central glossary, or a term to be modified for this PI.
- *References* would be next and provide references for higher-level PIs, lower-level PIs and PIs at the same level as this PI that a person may need to know about to understand it. As needed, references include textbooks, forms, etc.
- *Dependencies* follow and identify any activities, products, or services outside the control of this process, and not covered in the reference section, to which the success of this process is linked. For instance, if a PI required a template, the template's path name would be provided. If another PI needed to be implemented, that PI would be identified here.
- The *participants* section created discussion about what the paragraph title should be (e.g., roles, participants) and the need for this section if the responsible roles are identified in the procedure steps. The PM's preferred term, participants, was used. This section identifies personnel by title; organizations or groups participating in the given process; a brief description, if not already in the Process Dictionary; and a list of duties/roles for the PI. The purpose of this section is to provide an overview of roles and responsibilities.
- PI diagrams *(process mappings)* were a management requirement. A kind of diagram-style chart was agreed upon. The basic format would be input, process, and output (going top to bottom of a page). Each box, decision diamond, etc., would include a number(s) pointing to the related procedure step(s), identifying the participant for the action, and the action to be performed. If a diagram was to take several pages, it could be moved to an appendix. One person mentioned that diagrams are needed since, "A picture is worth a thousand words." A counter statement was, "But which thousand words?" A diagram must have the same meaning for everyone and clearly represent the intent of the text. PI diagrams are now required for all drafts.
- *Procedure steps* would be a numbered list of action steps, with one or more participants. The major discussion was to organize the steps along a time line or by each participant. We decided to use a time line, especially since the participant paragraph summarizes the participants' roles. A short discussion on the need to number procedure steps ended when the following illustration was given: Without numbering, how does a step refer to a previous or later step? During development of the first PI, a situation arose where a step was a lead-in sentence to a list (e.g., the statement "Identify topics") followed by some substeps. The agreement was that in this type of situation the substeps would be separately numbered procedure steps, but indented. It turned out that this became a popular PI style.
- Using checklists brought up various discussions, e.g., use table formats (which may have a psychological image of limiting checklist comments). Most people preferred not having a PI checklist. The issue was discussed as to how to show each process has been followed; for example, is a PI necessary to show that a PI has been properly implemented. The final decision was to use a checklist only when needed ("when needed" was not defined). As of this writing, this section was known as results and only one PI had an entry (a form to report peer review comments).
- As part of our process, required forms had to be attached to the back of the PI. An exception to this was the use of a reference if a form was part of another PI. If a form was a template, the template would not be part of an appendix, but identified in the reference section with its path name.
- Because some PIs overlap organizational lines, we discussed the need for management-level PIs. These would address processes not covered by lower-level organizations or that crossed organizations. In our situation, we saw no need for management-level PIs since the only formal prime-contractor company relationship we had was financial, contractual, and status reporting. If needed, the process owner could be the PM. This decision reduced the number of needed PIs.

Prioritizing the Processes

Since agreement had been reached on PI format and style, the next issue was the order in which PIs needed to be developed. Each organization developed a list of possible PIs and the organization's recommended priority. From this the PDG agreed on an IV&V priority list. A schedule was developed, along with a list of PI authors from the PDG. It was also agreed that the schedule would be a guideline rather than a mandatory requirement.

Process Instructions (PIs)

After the administrative details were worked out, the PDG started writing the PIs. The first PI was peer review. Part of the main discussion was:

- Will this be a peer review of documents we developed or a peer review of the comments on the documents that our customer wanted us to review?
- The answer to the above question would also address whether "the peers would be people with the same knowledge as the author, or a real management review."

For this PI, the peer review process would be the review of IV&V-generated documents by people with the same knowledge as the author. Similarities of this PI to a future PI for the review of external documents would be addressed when the review of externally generated documents became the focus.

Process for Creating PIs

After the PI format, schedule, etc., had been agreed upon, the general PI development process was:

1. The PDG tasks someone (e.g., an author) to write a PI.
2. The PI is submitted to the PDG for review.
3. The PDG convenes, discusses changes, and generates comments.
4. The author makes designated modifications and resubmits the PI to the PDG.
5. The author distributes the draft PI to the organizational leads for their comments.
6. After the author addresses the organization leads' comments, the PDG addresses these changes.
7. Upon completing its work, the PDG distributes it to all project members for their comments.
8. The PDG and the author address each project member's comment. If needed, the PI returns to the project members for more comments.
9. Upon PDG approval, the PI goes though a coordination process (i.e., signing by the Process Owner) and goes to the PM for final approval.
10. The approved PI goes to the IV&V librarian for incorporation into the project library.
11. The IVV librarian notifies the project members of the new or revised PI.

Process Improvement

About two weeks after the first PI (peer reviews) was approved and implemented, the author of the second PI (a status report) ran into problems. The Peer Review PI did not account for the customer providing inputs to the draft documents under review. Allowing for the parallel processing of Peer Review and customer comments was seen as causing problems. Receiving customer comments and initiating the Peer Review PI would cause too much of a delay, i.e., we could not meet our deadline. The Peer Review PI also assumed it would initiate the distribution of documents to the customer. The second PI needed the author of the document to initiate distribution of the status report, again due to customer requirements not addressed by the Peer Review PI. Thus, the scope of the Peer Review PI had to be reduced, which resulted in rewriting the Peer Review PI.

This problem with the Peer Review PI also made the Process Development Group rethink the identification of PIs to be developed and how they would be developed. It was decided to develop a scheme to introduce new/modified business activities and how to associate these activities into PIs. As of this writing, this process had not been finalized, but an approach was developed and is awaiting approval.

Conclusion

The Mars Climate Orbiter (MCO) spacecraft, the first interplanetary weather satellite, failed on September 23, 1999 due, in part, to one group using metrics and another group using the English measurement system per the National Aeronautics and Space Administration (NASA). As stated by NASA,

> *". . . sufficient processes are usually in place on projects to catch these mistakes before they become critical to mission success. Unfortunately for MCO, the root cause was not caught by the processes in place in the MCO project"* [2].

Besides the root cause, eight contributing causes were also identified, including [3]:

- The system engineering process did not adequately address transition from development to operations.
- Inadequate communications between operations and project management. This is illustrated by the following: *"Although the navigators [those who controlled MOC's approach to Mars] continued to express concern about the spacecraft trajectory, NASA's [Arthur] Stephenson explained why there had been no concern shown by management. "[Navigators] did not use the existing formal process for such concern," he stated. JPL [Jet Propulsion Laboratory] has a special form to invoke a so-called incident surprise and analysis procedure, and the navigators did not follow the rules about filling out that form to document their concerns"* [4].
- The Verification and Validation process did not adequately address ground software.

Even though this paper provides an example of how a group began developing its best practices, the above illustrates the importance of ensuring that interfaces between groups are addressed and implemented. The work to document best practices has little value if the follow through is totally or partially ignored.

Remember: Best practices do not necessarily result in good products or services.◆

References

1. Humphrey, Watts S., *A Discipline for Software Engineering*, Addison-Wesley Publishing Co., Reading, Mass., 1995.
2. *NASA News Release:* 99-134, Mars Climate Orbiter Failure Board Releases Report, Numerous NASA Actions Underway in Response, Nov. 10, 1999.
3. NASA, *Mars Climate Orbiter Mishap Investigation Board Phase I Report*, Nov. 10, 1999.
4. Oberg, James, Why the Mars Probe Went Off Course, *IEEE Spectrum*, December 1999.

About the Author

George Jackelen works for EWA Inc., and has many years of experience in the Air Force and industry, having worked all aspects of systems and software life cycles. He has a bachelor's degree in mathematics from St. Cloud University and a master's degree in computer science from Texas A&M University.

1000 Technology Drive
Fairmont, W.Va. 26554
Voice: 304-367-8252
Fax: 304-367-0775
E-mail: gjackele@ewa.com

Risk Management Fundamentals in Software Development

By George Holt
Materials, Communication and Computers Inc.

Software development and engineering is a rewarding endeavor, but not without risks and challenges. Efficiently managing these risks is critical and requires that all parties, including researchers, engineers, developers, programmers, and project managers keep the fundamentals of sound application development top-of-mind. Staying focused on the basics is an essential part of minimizing risks and ensuring the success of even the most challenging and complex development projects. Periodic review of these principles and practices is valuable for even the most experienced developer.

The Challenge

Developing software is seldom an easy task. Each project entails unique demands, challenges, and problems. Failure to predict and prevent risks can lead to costly delays, revenue loss, increased stress on team members, a lesser product—even project failure.

The Solution

Although each project has its own requirements, the characteristics of effective risk management remain the same. By identifying risks and developing solutions before and during the development process, you maximize the team's efficiency and the quality of the finished product.

First Line of Defense: Strong Fundamentals

Careful evaluation of potential risks, and solutions to address them, is only the first step. Recognizing that sound application development principles are central to effective risk management and that the risk-management process is ongoing is the key to maximizing the team's efficiency and ability to create excellent software.

Flexible Planning

First, remain flexible. Giving team members the freedom to make decisions as new information becomes available enables them to react quickly; inflexible plans and schedules impede the ability to deal with new challenges.

Inflexible schedules also present team members with a tainted view of the project's progress. While plans and schedules are necessary to measure progress and meet deadlines, they cannot be too rigid. Promoting flexibility in the initial stages of the project encourages proactivity, positions the team to meet the challenges that undoubtedly arise, and also makes them better prepared to maintain realistic schedules as the project progresses.

The importance of flexible planning is particularly important during the creation of the software development plan—the road map that guides the team's efforts.

Sound Development Principles

During the development phase, important unknowns critical to success will unfold as the project proceeds. These unknowns are often risks. The key to accomodating these unknown factors is to recognize that you cannot know everything that may happen. It is important to breed a healthy respect for lucid ignorance, particularly at the beginning of a new project and to guard against individuals who pretend they know all of the answers.

It is often tempting to place too much emphasis on pseudo-order to balance this uncertainty. Reports are completed, meetings are attended, and schedules are met, but few realize how much progress is being made or what risks are increasing in scope. Recognize that the final goal of software development and risk management is excellent software. Do not get so wrapped up in reports and schedules that they become the key area of concern.

One key sound development principle is to reduce the complexity of your code and to modularize common functions. Segregate code modules that will not be shared with all target platforms or operating systems. Make sure the remaining code is common. This optimizes your efforts and lays the groundwork for easy fixes down the road as well as efficient reuse of software on future products.

Integrated Product Teams (IPTs)

Forming IPTs is another valuable approach to containing costs and reducing risks, especially those that might effect scheduling. The IPT facilitates problem solving, enables the team to rapidly respond to changing requirements, and prompts everyone to work on schedule.

IPTs are also an excellent way to keep the customer, in most cases the government, up-to-date on how changing requirements will effect the cost and scheduling of a project, or present the team with additional risks to consider. This is particularly true if the customer plans to add additional features. Most importantly it helps you, the developer, appraise risks and provide acceptable solutions as the customer raises questions.

Prototyping

Exploratory prototyping is the first step toward avoiding the costs of full research and development. It also is an excellent strategy if project requirements are ill defined or likely to change before project completion. In addition, exploratory prototyping is an excellent way to clarify requirements, identify desirable features of the target system, and promote the discussion of alternative solutions.

Prototyping should answer two questions that are fundamental to software development and risk management—"Is the concept sound?" and "Is it worth proceeding further?" If the answer is not a clear "yes," you may be setting yourself up for failure. More importantly, without this insight, you will give the customer a false sense of what can be accomplished. Better to know this up front. Sometimes the most important risk management action you will take is to ask these fundamental questions.

If the answers and the risks are satisfactory, you can move on to evolutionary prototyping, which offers several benefits. It enables your team to quickly and efficiently build on proven aspects of the software. In addition, it enables end users to better define the remaining requirements. As a result, the core of the software's foundation is tested and proven early in the project, significantly reducing exposure to unknowns.

Process Improvement

Improving processes should be ongoing throughout the project. It is important to continually ask, "Is there a better way to get the job done?" Improving the way you do things cannot be done in a vacuum—communication at all levels is critical. Participate with your customers in IPTs and system management teams. In addition, be sure to meet with the teams' engineers on a regular basis for focused, but informal, discussions. While these meetings are exceptionally valuable, guard against extended meetings that cut into your teams' work time.

One alternative to lengthy meetings is to develop and distribute weekly status reports. These give each member insight into the progress of the entire project and a clear view of the big picture. Remember that you can have the best processes in place and still fail miserably in software development. A motivated, goal-oriented, and knowledgeable workforce will succeed even when the process is lacking.

Quality Management

The best quality management approaches use empowerment, ownership, and consensus to minimize risks and maximize productivity of the project's workforce.

Empowering your team members enables them to be their best. All development teams are concerned with giving their engineers the tools they need to succeed, but many fail to provide them with the freedom to use them. Even the most prepared project will suffer if the team is bound by extensive rules and regulations. Do not stifle initiative and creativity.

Always allow the team to propose solutions or actions without the fear of undue punishment and ridicule. Empowering the members of your team improves efficiency, encourages exchange of ideas, and increases the intellectual capacity of the group. In the end, the customer reaps the benefits of this increased expertise.

As in all things, ownership is a key component of success. People care more about the things they own, and software development projects are no exception. Fostering a team-wide sense of ownership makes each member accountable for the success of the project. In addition, members who feel they own their work are much more receptive to constructive critique of their decisions.

Developing consensus at the beginning of a project is also a key aspect of quality management. From the beginning, allocate sufficient time for everyone to agree on the functionality of the end product and the development of the theme. The theme should describe the purpose of the product rather than technical details.

Teams that lack this consensus and a clear-cut theme for the software often place too much emphasis on additional features and technical aspects. These features, although nice to have, usually do not contribute to the basic functionality of the software and tend to lengthen the development process. A good theme guides the order of development and provides the team with a focus that is supported by all involved.

Third Parties: A Mixed Blessing

If a product does fail, it is common for many developers to blame the project's failure on third parties. In some cases they are correct. At times you will have no choice but to elicit their help. The key is to minimize how much you depend on them.

Anytime you rely on a product or service from someone outside of your group your risk of failure or delay increases. Your team may do everything right, but if a crucial third party does not, your work may be in vain. To illustrate this, consider the risks you assume by depending on three crucial components of your project from start to finish. Assume each product has an 80 percent chance of arriving on time and fully functional. The probability of success for all three combined is not 80 percent, it is .80 x .80, x .80 or 51 percent. In other words, your project now has a 50-50 chance of failing.

Implementing Effective Risk Management

Now that we have reviewed the first line of defense against risks in the development process, let us look closer at risk management techniques you can use in your own projects.

An effective risk-management program is dynamic and ongoing throughout the development process and requires the participation of everyone involved. First, remember it is an idealized process. Do not follow it lockstep for each and every development undertaking. Modify the process depending on the type of work to be performed and the members of your team. For example, software rehosting or software block updates may not require the same degree of discipline as a new development effort.

To implement an effective risk-management program requires careful review of risk assessment and risk control.

Be Prepared: Risk Assessment

Carefully assessing challenges inherent in any project is the first step in implementing a successful risk-management program. Risk assessment includes three key processes: risk identification, risk analysis, and risk prioritization. Focusing on each ensures the successful application of risk control tactics.

Risk Identification

Identify the risks most likely to occur in the project and pay particular attention to those created by changes in customer requirements or target systems. Develop a checklist of the risks that may arise and include input from each team member.

Risk Analysis

Once you know what risks you face, it is important to analyze each one in two ways; first, the chance of the risk occurring and second, the consequences if it does. Looking at each risk and answering each question enables you to determine which risks require focused attention. Modeling techniques can be helpful in determining the odds of each risk occurring, but each team member's input is essential in determining potential consequences.

Risk Prioritization

Developing a plan and determining which risks you need to deal with first is critical. Prioritize your risks based on your analysis of the risk's chance of occurring and the potential consequences. Risks that pose the greatest danger to the project must be dealt with first.

Take Action: Risk Control

To implement a successful risk management effort requires continuous assessment. Risk control includes three key processes: risk management planning, risk

Project managers and developers should collect and monitor metrics that give a picture of the project's progress. If the results of these metrics suggest that a change in process is required, the development practices will need to be modified. Some metrics that pay big dividends are:

REVIC/COCOMO II: These models estimate the cost of software projects and have very good predictive capabilities if environmental factors are carefully estimated and applied to the models.

Schedule Adherence Measures: Use this metric to plot the progress of the project against your planned progress for all major and subtasks.

McCabe's Cyclomatic Complexity Metric: This tool measures the complexity of single-code modules.

Fan Out: This model measures system or structural intermodule complexity. Measuring such factors as fan out, or the number of modules called by any given module, will show the direct correlation of overall system complexity and development defect rate.

Software Cleanroom Process: The cleanroom software process stresses proof of correctness in the design and coding phase of development.

Readiness, Maturity, Growth Model: This model provides management with a quick look at the state of the software development effort. It is very useful to determine when software is ready for formal qualification testing.

Curvilinear Relationship between Defect Rate and Module Size: This metric provides the optimum size of modules to minimize overall defects.

Early Defect Removal: Software engineers should concentrate heavily on early defect removal in the design and coding phases of software development. It has a substantial impact on reducing program costs. It is also important during development to conduct informal analyses to eliminate processes that cause errors.

SW Error/Fix Ratio and Defect Removal Efficiency: This measure is used to gain insight into the software team's proficiency in resolving errors, or bugs, in a timely fashion.

Software Lines of Code (SLOC) Tracking by Functional Builds: Compares the actual vs. estimated SLOC per build.

resolution, and risk monitoring.

Risk Management Planning

Risk-management planning is based on the likelihood and consequences of a risk occurring as determined by risk analysis in the assessment phase. Reacting to risk requires resources and time. Always evaluate whether the costs incurred by implementing risk control outweigh the benefits. This enables the team to plan its risk management efforts rather than respond to each risk as it occurs.

Risk Resolution

Once you identify the risks in a development project, the next step is to resolve or reduce them. Employ staffing decisions, cost/schedule estimates, quality monitoring, new technology evaluation, prototyping, scrubbing requirements, benchmarking, and simulation/modeling to react to these risks. Remember, in most projects, 20 percent of identified risks account for 80 percent of potential for project failure.

Focus on the 20 percent with the most likelihood of causing problems. Employ early prototyping and frequent functional system builds to determine if the steps you took to resolve the risks were successful.

Risk Monitoring

It is important to continually monitor your risk control efforts and the appearance of new risks as a result of prior fixes. As the key to your risk monitoring efforts, insist that the program manager, the technical lead, and each developer are able to state their top three risks at any time. These are by their nature dynamic and will change.

Risk Tracking Tool: Risk Radar© is a useful tool to track, manage, highlight, and contain risks, especially those that lay on a critical path. It does not attempt to replace professional judgment, but it is a straightforward, easy-to-use tool for tracking, prioritizing, and communicating project risks. It focuses on ease of use by displaying important, fundamental risk data and provides a graphical display of risks by probability of exposure and has extensive reporting features. It is free and can be found at www.spmn.com/rsktrkr.html

Summary

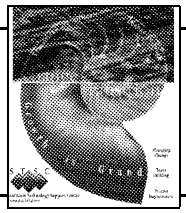
Software development will always include risks, but none are insurmountable if you are prepared to face them at the start. Risk management is an excellent way to prepare for daily challenges.

A viable risk management plan can mean the difference between success and failure. It should, above all else, be flexible and encourage initiative. Remember to always look ahead, use rapid prototyping if necessary, follow a defined program to minimize and manage risks, use a good set of metrics, keep the customer in the loop, and always follow the fundamentals of sound application development. Following this risk management approach will not guarantee excellent software

About the Author

George Holt is a vice president of Systems Development Division of MATCOM Inc. He has developed software for more than 15 years, ranging from military tactical systems to simulators to computer-based tutorials. His division recently rehosted 180,000 lines of Army tactical software to a nonproprietary, open-architecture, Pentium-based system for which it received the Department of Defense Standardization Award. He also managed the Digital Fire Control System prototype for the Lightweight Howitzer, which was successfully fielded in six months. He is the author of many magazine articles, technical publications, and co-author of the book, *Strategy: A Reader.*

1050 Waltham St.
Lexington, Mass. 02421
Voice: 781-862-3390
Fax: 781-402-1515
E-mail: gholt@lexma.meitech.com



Transition is difficult . . . Change is tough . . . Growth is Grand . . . .

With this in mind, CROSSTALK would like to thank Larry W. Smith for designing the back cover ad on this and the previous three issues (in addition to the current and several previous covers).

Development of Space Shuttle Telemetry Station Software

by Dr. David K. Mann
United Space Alliance

This paper reports on the space shuttle telemetry ground station development project turnaround brought about through collaborative software development and sustaining engineering. The application of this new project and software development methodology to the development resulted in several positive effects over the standard development method. These include a reduction in the initial software development costs, a reduction in the software time to market, improved marketability of the software technology developed, improved product quality, improved maintainability, and technology transfer. This collaboration between the United Space Alliance (USA) telemetry station development team and APLabs Inc., resulted in software innovations in Frequency Modulation (FM) and Pulse Code Modulation (PCM) processing software, as well as station management software.

Daniel Golden, NASA Administrator, in his strategic outlook for 1999 [1] provides a statement of strategic intent for the agency. In this statement he outlines a three-part mission in which "Technology Development and Transfer" is a cornerstone of the mission for the agency in 2000. This is consistent with the 1999 external assessment [2] in which the administration places priority on the promotion of "high technology for economic growth through effective partnerships." The prime Space Flight Operations Contract, SFOC (reference: Contract NAS 9-20000) awarded to United Space Alliance, section G-14 and G-15, requires the contractor to provide a portion of the contract funds to small business and to support the "Government's Technology Transfer Program." This new development method is an example of how USA is exploring new ways to increase the marketability of technology developed on the space program while decreasing Space Shuttle Program development and operating costs.

This new collaborative development method involves adopting a software application counterpart available in industry to provide baseline functionality and developing additional capabilities required for the upgrade or replacement system in collaboration with the software vendor. Applying this new development method more effectively leveraged technology and expertise available in industry to reduce initial software acquisition cost and time to market, while providing a superior product to space shuttle operations. Also, the technology developed is built on the state-of-the-art rather than reinventing the state-of-the-art, making the technology developed more valuable to industry and the American public.

The project defined a software development turnaround in terms of key project management metrics (i.e. cost, schedule, and technical). Turnaround was defined as a 50 percent reduction in anticipated software development labor and time to market. The improvement in the technical merit was a little harder to quantify. Two surveys were performed as part of a comparative analysis. Three key technical categories were defined and a turnaround was defined as a marked improvement in two of the three. The categories were marketability, software product quality, and maintainability. Attributes were defined within each category. The first survey was designed to determine the importance of the attributes within each category. The second survey was performed after software development was completed and scored the delivered product against the most likely outcome of continued development on the custom code. The results of these surveys were organized and presented in a Kepner Tregoe decision matrix for comparative analysis [3]. A marked improvement was defined as 100 percent improvement in absolute score within a category.

Development Method Genesis and Vendor Selection Process

A detailed estimate to complete the project, assuming continued status quo software development was performed, provides a baseline for evaluating improved project performance after the change in direction. This estimate was based on a functional analysis of the custom code under development against the functional requirements for the upgrade.

Functionality was divided into three categories for this analysis: telemetric, station management, and data products and tools. Telemetric functionality was defined as the capabilities to acquire measurement data from the PCM or FM carriers, route this data to a location on the ground station, and capture the data in a file for production of data products. Station management functionality was defined as the capability to set up and manage data acquisition as well as monitor real-time elements status. Data product and tools were defined as the capability to process the raw data acquired into data products.

Figure 1 is a summary of these analyses. The entire pie in each category represents the functionality required for the upgrade in each category. The three slivers contained in the "Functionality Addressed in the Custom Software" regions represent capability outlined in the custom code. The two black slivers in this region represent the capabilities inherent in the first two releases of the custom code. The third shaded slice in this region represents outlined but not functional capabilities. The slices remaining outside the "Functionality Addressed in Custom Software" region represents functionality required by the Shuttle Telemetry Station that was not anticipated during the custom development to date. The analysis revealed that approximately one-third of the telemetric, three-quarters of the station management, and two-thirds of the data products and tools capabilities required new development assuming continued status quo software development. Project leadership and management derived an estimate to complete for the custom software. This estimate was based on a detailed knowledge of the functionality required (function point analysis), performance histories of the developers involved, and developers' estimates. It was estimated that completing the custom code would require an additional 20 man-years of productive software development effort over five years

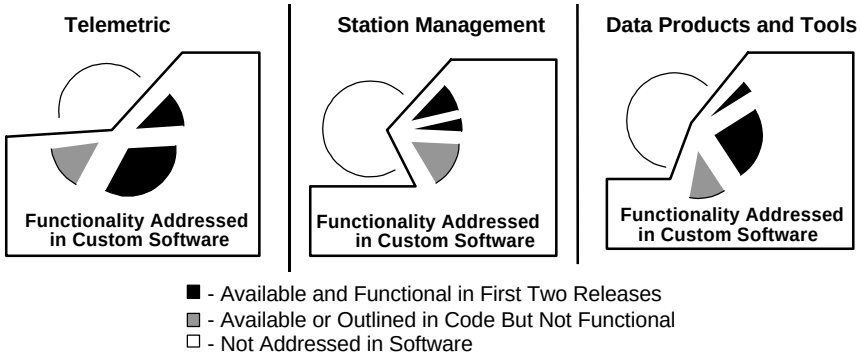
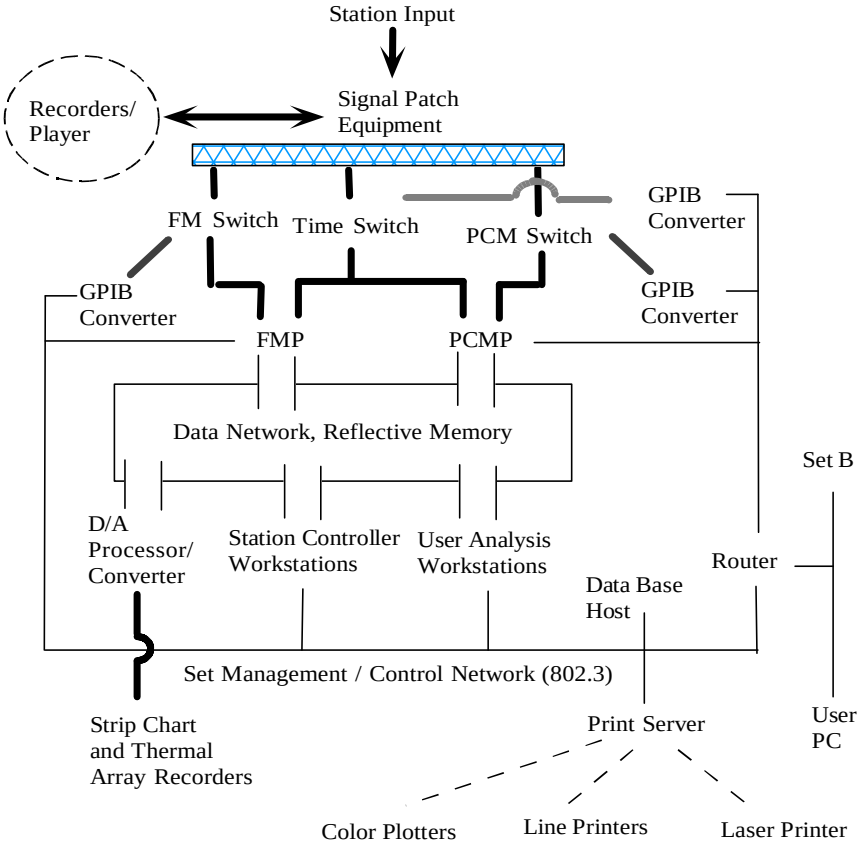


Figure 1. *Evaluation of the Custom Code Revealed Large Gaps in Functionality*

with a probable growth of 20 percent in manpower and schedule.

An industry survey, coupled with a series of product evaluations, was performed to determine if a commercial off-the-shelf (COTS) product that met the functional requirements for the upgrade was available. The survey revealed that no COTS package met the upgrade requirements. There were, however, several close functional counterparts—Veda Systems, Huerikon, Harris, Metraplex, APLabs Inc., Avtec and Acromag. Hardware for the upgrade project had been purchased and major change to the hardware architecture was cost prohibitive. As product and vendor investigations progressed, it became apparent that general-purpose telemetry software packages were coupled to particular hardware architectures. Of course, anyone can do a port, but the challenges associated with developing additional capabilities concurrent with a port to new hardware architecture was an approach deemed to have excessive risk. In addition, the vendor selected had to provide development and licensing flexibility, which will be discussed later. These factors lead to a single viable vendor and product combination. APLabs Inc. wrote VMEwindow to a hardware architecture similar to the one that was in storage for the upgrade and provided the flexibility to complete the project. A functionality assessment, similar to the one performed above, was done in order to scope the software development effort required and justifies the change in strategic direction to management.

Figure 2. *Ground Station Overview*



Collaborative Development

Collaborative software development started the second week of March 1998. The first order of business was to train USA developers in the VMEwindow development environment. Systems engineering was performed to identify the system level requirements and identify gaps in functionality between the baseline VMEwindow product and the required upgrade ground station. Although there were many minor modifications, the collaboration resulted in four software products, with innovators from USA and APLabs Inc.

These were submitted to NASA for Technology Transfer. Software development was completed the second week in November 1998. These software products represent modification to existing technology to improve and expand the capabilities of the baseline product. Figure 2 is a block diagram of the upgrade ground station architecture and is referenced in the following discussions on the software developed. The hardware architecture consists of Sun workstations connected to multiple PCM and FM processing VME chassis that use reflective memory as a real-time data transport mechanism. The chassis use Motorola processors and SBS Berg telemetry System cards. The software architecture consists of the VxWorks operating system, VMEwindow, Matlab, Dataviews, PVWave. The software products developed for the project and submitted to NASA for technology transfer are described below.

Sbus SCRAMnet Interface for VMEwindow

A reflective memory network shown in the center of Figure 2 is the primary data transport vehicle for raw data from the PCM Processors (PCMPs) to the Digital to Analog Programmable Converters, User and Station Controller Workstations. The VMEwindow ground station telemetry package did not support a SBus Shared Common RAM Network (SCRAMnet) interface required to get data to the workstations. The challenge was to develop an application using the VMEwindow development environment that would provide the capability to

acquire and display data at a single SBus workstation from multiple real-time telemetry processors in a deterministic manner with minimal latency. The code needed to be integrated within the VMEwindow environment, providing the operator with a consistent interface. It was decided that a derivative work could be produced from the code already available for interface to the VME version of the SCRAMnet card. It was negotiated that AP Data System would perform code modification and initial development testing and USA would provide the reflective memory topology, memory offset and data requirements in addition to concurrent code review, integration, and testing.

Stream Definition Flat File Import Capability for VMEwindows

This software provides the capability to automatically load telemetry stream definition. This capability was seen as critical to the design because of the thousands of measurements that must be loaded to support each mission. Space Shuttle PCM downlink, data location, and unit information is contained in a database. Baseline VMEwindow provides a manual utility to load this information into a stream definition but did not provide the capability to automatically load stream definition. The software product developed provides the capability to create an ASCII flat file from a SQL database and import this flat file into the VMEwindows stream definition. This capability not only reduces time required to set up to support a mission, but also improves the reliability of the software load by cutting down on input errors. The user is prompted for parameters to define a PCM stream and accepts lists of measurement names read from an existing file and accesses the database to create an ASCII flat file. The generated file contains all of the information necessary to define PCM stream and populate the stream definition in VMEwindows.

GPIB Board Setup and Control for FM Snapshots and Calibration

This software provides the capability to set up and control the NI1014 General Purpose Interface Bus (GPIB) board for specialized FM functions. Although a generic GPIB interface capability was available in the baseline VMEwindow environment, it did not meet the requirements of the project. The system uses a Metraplex digital discriminator and a Keithly switcher. The setup and control software allows individual control of both the discriminator and the switch and automated control of report generation. The setup functions were integrated into the ground station software as a newly developed VMEwindow icon. Board level control is provided using existing APLabs Inc. driver software. The software provides a calibration and FM Snapshot capabilities. A FM test signal that shifts the frequency over the bandwidth to represent five levels between -5V to 5V is input to the digital discriminator. The calibration software provides an average of five samples at each of these data levels. Once the set is calibrated, a snapshot can be produced providing the average, minimum, and maximum values for 100 samples of real-time data.

Datel 622 Digital-to-Analog Driver, Setup, Control

This software product provides the ability to produce thermal array charts of data with loss of signal event indicators. The driver and control software provides the capability to set up and control the digital-to-analog conversion functions of the board. This capability was not available in the baseline VMEwindow environment. The setup functions were integrated into the groundstation software as a newly developed VMEwindow icon. The control software provides the capability to select data channels to be output, select event indicators for edge trace output, scale processing, and calibration functions.

Collaborative Support, Development, and Licensing Agreement

There are several challenges facing successful application of a collaborative software development on a project in support of shuttle operations. Some challenges stem from the perception that the ability to support operations is somehow compromised. Others are from the notion that derivative or new software technology developed adjunct to a baseline application is not transferable to NASA and industry. Another argument in favor of in-house custom software development is that developers often are most qualified to troubleshoot and repair time-critical bugs during processing. Additionally, if the vendor was to go out of business or drop the product, the space program would run the risk of losing support for operational software. The project had to ensure that each of these issues was addressed and equal or superior capability would be available after the upgrade was complete.

These issues were addressed in several ways. First, in-house software developers were trained and certified as VMEwindow developers. This provided us the capability to collaborate on the development of required new capabilities and sustain the software after the upgrade was complete. Second, we negotiated a "Collaborative Support, Development and Licensing Agreement" with the vendor. This agreement has several clauses designed to address the concerns above. Rights to the source code for the purposes of Research and Professional Services' shuttle processing and technology transfer to NASA were secured in a "Project Buyout License" clause. This eliminated the need to escrow source code with a third party and enabled ad hoc modification of the baseline product during mission support without infringement concerns. Acquiring special user and software developer support, ensuring that immediate attention is paid to software issues during critical processing times in collaboration with in-house developers, addressed operational support concerns. A product upgrade cycle is defined in the agreement where new version software is provided, along with user documentation and installation assistance. This ensures that we have the ability to stay current while minimizing configuration management costs. The software development collaborations were so successful that special provisions were negotiated to ensure future endeavors would adopt a similar course. Subsequently, two new software innovations have been developed and will be written for technology transfer.

Findings

Software development cost savings are derived by comparing the actual resources expended to the estimated resources required to complete the custom code, assuming continued status quo development. The cost associated with purchasing the

COTS software and development services is conservatively converted to equivalent manpower and added to the actual in-house labor expended to complete the software. The resulting figure is 6.6 man-years; when compared to the estimate to complete the custom code, the figure shows savings of 13.4 to 17.4 man-years. Converting this to equivalent dollars shows more than $800 thousand to $1 million savings.

The software development was complete the second week in November 1998. This meant that the software development took eight months rather than the five to six years estimated to complete the custom code. This represents an 86 to 89 percent reduction in the anticipated software development time to market.

A comparative analysis was performed, designed to determine the relative technical merit of the delivered software to the most probable product assuming continued work on the custom code. Expert engineering judgment is relied upon as the basis for this analysis through two surveys. The first column of Table 1 lists the categories defined to represent technical merit for the purposes of this analysis. Subindentures under each of the three categories (i.e. Marketability, Software Product Quality and Maintainability) are category attributes. All responses were normalized, averaged, and reported on a 1-to-10 scale where 10 was the most important or best. The second column represents the results of the first survey. This survey was designed to determine the relative importance of the attributes within a category from the perspective of the management. The third and fourth columns represent the results from the second survey.

The respondents scored the delivered software and the most probable outcome of continued status quo development. The additional costs associated with COTS software procurement and development was converted to equivalent manpower and the respondents were asked to estimate, assuming these additional resources were brought to bear on the custom software development. The fifth and sixth columns list the absolute scores for the custom and delivered product, respectively. These figures are totaled for each category to obtain an absolute relative score for each of the alternative methods. Comparing absolute scores reveals a three- to four-fold improvement in absolute score in every category.

Table 1. *Method Comparison Matrix*

1	2 Average Normalized Weight	3 Average Probable Custom Product Score	4 Average Software Product Delivered Score	5 Absolute Score Probable Custom Product	6 Absolute Score of Software Delivered
Marketability					
Value to Current Customer	7.78	1.67	9.33	13	73
Value to Other NASA Projects	4.70	2.50	8.50	12	40
Value to Industry	2.22	2.67	8.33	6	18
Value to Future Shuttle Customers	6.38	2.00	9.00	13	57
Value to American Public	3.92	4.17	6.83	16	27
Total =				60	215
Software Product Quality					
Usability	5.07	2.67	8.33	14	42
Reliability	8.26	1.83	9.17	15	76
Functionality and Versatility	4.57	1.83	9.17	8	42
Extensibility	2.09	1.50	9.50	3	20
Total =				40	180
Maintainability					
Training Programs	5.24	2.33	8.67	12	45
Availability of Trained Personnel	3.69	2.83	8.17	10	30
Ability to Enhance Software	5.55	1.67	9.33	9	52
Documentation	3.42	2.00	9.00	7	31
Configuration Management	7.09	3.58	7.42	25	53
Total =				64	211

Conclusions

Comparing the actual project performance and the projected performance, assuming continued development of the custom code, reveals that cost and schedule were reduced. The technical evaluation of the product delivered revealed a marked improvement in every attribute within all three categories evaluated. Several additional benefits included technology transfer, continued collaborative enhancement of the product, and the ability to combat obsolescence.

This methodology represents significant improvement over the status quo and should be evaluated for implementation on future and ongoing NASA software development projects. It is the author's considered opinion that the results warrant application of this methodology where close counterparts are available in industry. Strong close technical counterparts for data warehousing, recording, retrieval, command, control, data monitoring, network traffic generation, and system administration functions can be found in industry.◆

Acknowledgments

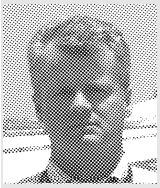
The author gratefully acknowledges Jonathan Morsics, Steven Prenger, Steve Gills, Andy Mason, Richard Price, Tom Perez, Earl Johnson, Bruce Chamberlain, the Integrated Data Systems Management Team and the Record and Playback Subsystem Upgrade development team.

This work was performed under the auspices of the Space Flight Operations Contract (SFOC, NAS 9-2000).

References

1. Reference: http://hq.nasa.gov/office/nsp/outlook.htm

About the Author



Dr. David Mann resides on Merritt Island, Fla. with his wife and two daughters. He works for United Space Alliance on space shuttle telemetry and computer systems as a project lead. He has 15 years of systems engineering, design, and project management expertise obtained on NASA and Department of Defense-related programs. Mann graduated with his engineering degree in 1985 and was recently awarded a doctorate in engineering management.

United Space Alliance
Launch Processing System Engineering
8550 Astronaut Blvd., Miss., USK-489
Cape Canaveral, Fla. 32920-4304
Voice: 407-861-7234
Fax: 407-861-7473
E-mail: David.k.mann@usago.ksc.nasa.gov

A New Application of CONOPS in Security Requirements Engineering

by Darwin Ammala
MPI Software Technology Inc.

The Concept of Operations (CONOPS) document, IEEE Standard 1362-1998, is a powerful tool for communicating a customer's vision for a new system. Normally used for describing full systems, CONOPS can also be used to address one single aspect—such as security in a large-scale project. This paper reports on a recent Navy contract effort, which demonstrated this use of the CONOPS. This paper will describe and analyze the results of this effort.

The IEEE CONOPS document [1] will gain in popularity, once the software engineering community discovers its flexibility, extensibility, and versatility. This paper summarizes a recent case of employing the CONOPS document to describe the desired multilevel security, and high-performance computing infrastructure characteristics to be included in the Navy's next generation combat vessels. The DD21 (21st Century) land attack destroyer is one such ship undergoing design competition. The Navy's Small Business Innovations Research solicitation topic [2] identified this vessel as a candidate for the most advanced multilevel security in a computer-intensive environment.

User Requirements

Traditionally, the Software Requirements Specification (SRS) has been developed as the primary document for stating user requirements. The SRS usually is produced by the developing organization following discussions with the potential user of the system and analysis of the requirements gleaned from these discussions. This document is well suited for use by its authors, but may be of less value to others, such as when users are presented the document for their comment. Users do not express their requirements at the level of specificity found in completed SRS documents.

An ideal SRS for a software-intensive system can be characterized as having requirements that completely capture a problem, is devoid of design evidence, and has succinctly stated requirements. This type of document would generally overwhelm the user in the sheer density of detail, repetitiveness of the language, and the total number of requirements stated. Most users would not be readily able to determine if the system described in this manner would truly address their original needs. The user's confidence in the development process can be fortified if he can see steady transitions of requirements from his expressed concepts, to requirements to specification, to code, to product.

> "With the new Unified Modeling Language now standardized, convergence of development methods is more likely. From a security, software, development perspective, this could improve the state of software security engineering."

People are most comfortable with describing things in the language of their problem domain. The primary objective of software engineering is to build the product that the user needs (validation); after this is to build that product correctly (verification). A high priority should be placed on allowing the user to directly express desires and ideas for needed capabilities of the expected system. The user may also provide some insight into specific testing criteria to be considered. Early attention to testing is also relevant during concept definition, as the system testing is a peer *off-core* development activity in the software development life cycle [3].

Security Requirements

A compounding requirement problem arises when security requirements are considered along with functional requirements. The area of Systems and Software Systems Security underwent growth during the 1990s. The Internet's emergence spurred numerous e-companies to rapid growth and success, while the field of electronic commerce remains in its infancy. Like all software, security-relevant software often is designed and released with latent defects. With the user community's heightened awareness of security, the developer community will observe a raised concern for sound, functional security within the software products that they are hired to produce.

Developers of these products employ disparate development life-cycle approaches, many of which are described by Hassan Gomaa[4] Barry Boehm[5], Alan M. Davis [6], et al. These authors do not single out security development practices per se, but one can infer that a definable development process is followed. With the new Unified Modeling Language now standardized, convergence of development methods is more likely. From a security, software, development perspective, this could improve the state of software security engineering.

Users, in most cases, have only rudimentary knowledge of security, and thus, are less likely to be able to help articulate their concerns in a language other than that of their native domain. The user is less likely to understand documents, such as an SRS, written by a developer community having security domain knowledge due to the specialized jargon. Terms such as *mandatory access control*, *nonrepudiation*, and *ubiquitous login capability* are far from everyday usage.

This leads us to conclude that perhaps security-relevant software engineering would benefit from a more thorough and well-understood collection of requirements. This would ensure that the right product is being built correctly. The CONOPS document written in user language is an ideal vehicle for this purpose.

CONOPS Overview

A CONOPS is a user-oriented document that describes system characteristics for a proposed system from the users' viewpoint [7]. The CONOPS document is written in order to communicate overall quantitative and qualitative system characteristics to the user, buyer, developer, and other organizational elements. It describes the existing system, operational policies, classes of users, interactions among users, and organizational objectives from an integrated systems point of view [1].

The CONOPS is intended to aid in requirement capture and communication

of need to the developing organization. Posing the problems to be solved in the user's language ensures that the user can more accurately express the problem. The developers then have a good basis to begin the requirements refinements, and initial design of the system.

Bringing different communities of people together (users in their domain, and software developers in theirs) implies that communication will be a critical issue. The IEEE CONOPS standard does not stipulate whether the user or the developer must write the document [7]. There are tradeoffs in every scenario—whether the user or developer writes the document. Users will better articulate the desired capability in terms of their domain situation, while developers are more likely to be familiar with current computing technology, and would express the required capability in terms of technology. [7].

Once a draft CONOPS is written, it is presented to the user and developer organizations for review and comment. Ideally, the user should write the CONOPS; however, the user must first be taught the intentions, and guidelines for doing so. Thus, collaboration between the user and developer is paramount until the user organization has mastered the techniques. The IEEE standard for the CONOPS document [1] does a good job of explaining the purpose for each section. Additionally, not all sections of the document are required. An agreement between the user and developer can establish sections that are needed, and which ones could be omitted. In practicing due diligence, each of the sections addresses unique aspects of the system life cycle and should be addressed.

A Security CONOPS

This paper now turns to the case study of the use of the CONOPS to articulate the security relevant portion of a proposed new ship-based computing facility.

Need and Solicitation

The overarching requirement came from an SBIR Phase 1 solicitation [2]:

Problem: The Navy's new ships require the most advanced technology and security services, adaptable to quickly changing conditions, and functional at new levels of efficiency while carrying fewer sailors to operate them.

Objective: To develop techniques to address multilevel security in a complex, software-intensive system. Of particular concern is maintaining multilevel security while supporting a robust ability to migrate and reallocate tasks through a complex computer network architecture [2].

Successful Proposal

The approach to this problem was to propose a business process review dialogue with the sponsor that would elicit the requirements—both computing system, and computing system security—to produce a security-specific CONOPS document. The anticipated interplay was for the developer to produce the initial draft of the CONOPS, and allow the user representative an opportunity to comment upon and edit the draft.

This approach rationalized that in light of the great complexity of a modern-day destroyer; the security should receive early, concentrated focus.

Delivery

The contract team met with the sponsors, and learned that this system effort was the first to employ a revised procurement procedure within the Navy. In prior procurements, the developer was given statements of work and statements of requirements that were refined and worked with the contractor to produce the requirements baseline documents, including the SRS.

The new procurement model employed by the Navy places more degrees of freedom on the developing organization to develop the product in the way it does best, while working with the sponsor. We learned that little information was available on specific DD21 requirements, primarily because the competition phase between the two project teams was under way and 18 months from completion. These factors gave us incentive to be creative in general computing capabilities, and concentrate on developing an adaptable computer-intensive environment that also employs sufficient security to meet mission requirements. This gave the CONOPS a technology-oriented composition.

The work from this contract served a bilateral purpose. The prime focus was to define security and system functionality requirements for the customer's new class of ships. The secondary focus was to initiate the users to the use of a specialized CONOPS document. Since the developer derived the requirements, the user's true requirements can be obtained only from the user community's review and modification of the document

Analysis

Computer-intensive environments, particularly those that must also provide multilevel security protection and services, are software-intensive systems. Building such software systems requires the proper employment of requirements engineering practices and tools. Among these practices are requirements elicitation and refinement. In cases where the developer has limited access to the users, the developer can offer a preliminary CONOPS document. This preliminary CONOPS will encourage the users to comment, clarify, or produce a response document, which is determined to reflect their views of what is needed. Once each side has had a chance to contribute to the requirements concept, a round of more probing analysis and questioning should follow. Employing an interviewing technique along the format described by Joseph Groguen and Charlotte Linde [8] to isolate areas of requirements that need more clarification would be an efficient use of time. This type of *zooming in* isolates a specific topic deemed critical in the new system, and allows detailed exploration of the problem and requirements related to it.

The security requirements were selected as the focus of initial operational concept. This was based on existing best practice in computing and information systems security that the system's security is designed prior to the production [9, 10]. Not doing so would entail retrofitting security into a complex software system, which if inadequately designed would be a precursor to failure. In the military domain, it is critical that security is implemented correctly and completely. This gets us back to understanding and capturing security requirements. Studies have proven that the developing organization's lack of understanding of the requirements is the

leading cause of project failure [3].

Our CONOPS document presents the general concepts for the type of computer-intensive control system that would be needed to meet the Navy's stated requirements for system resilience under dynamically changing conditions. The security CONOPS is the first document in the system's development life cycle. It is conceivable that many parts or subparts would have similar CONOPS documents written to describe them. The full system can be decomposed into mission or functional entities, each of which would have a CONOPS written to describe it, e.g., command and control, radar/sonar, fire control, general information technology support, etc. This full collection of CONOPS documents would represent the user requirements for the entire system. With this full set of documents, refining requirements and creating the System Requirements Specification, or Software Requirements Specification, could begin.

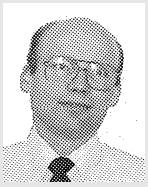
Conclusion

Complex software-intensive systems must have a thoroughly understood and soundly engineered set of requirements, which can be used along with best analysis practice to contribute to an effective design and ultimate implementation. The requirements are the foundation for the entire project, and must be understood precisely and managed diligently because changes are inevitable. It is advantageous to ensure that the real users of the systems are offered an opportunity to share their views and visions based on their working experience with the strengths, and weaknesses of their existing systems. Traditional software projects have been undertaken with less than complete understanding of requirements. The CONOPS document is a stride toward allowing the user's views to be heard, and allowing the developer to demonstrate to the users that their needs are understood and acknowledged.◆

References

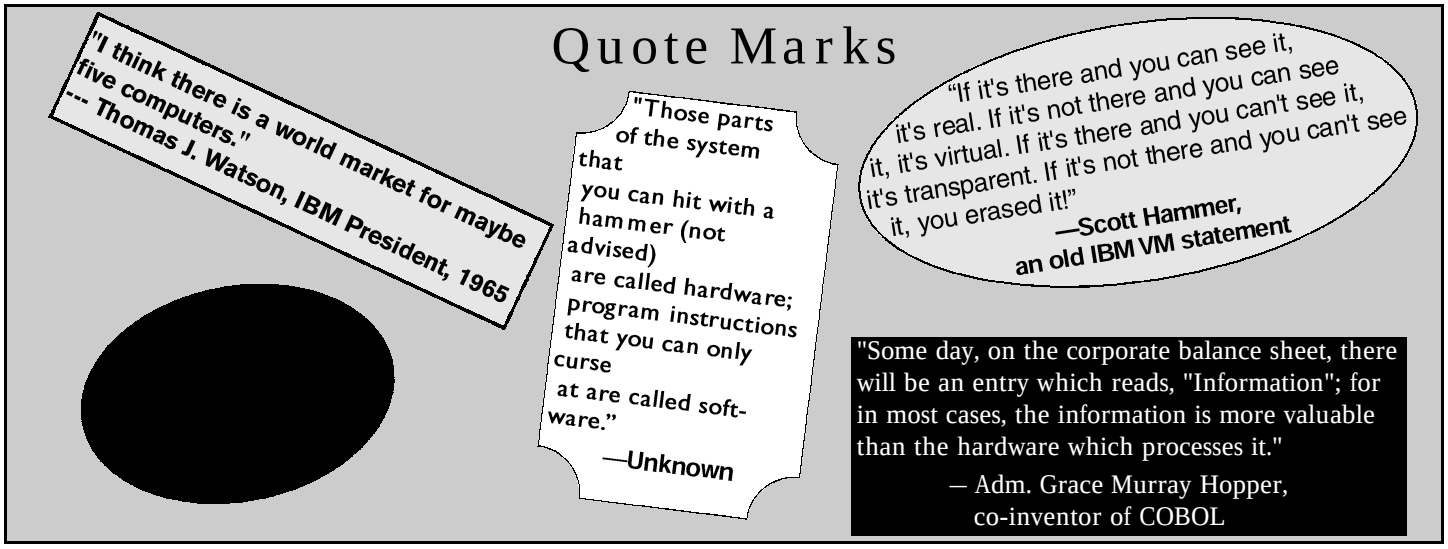
1. Institute for Electrical and Electronics Engineers, *IEEE Guide for Information Technology-System Definition-Concept of Operations (CONOPS) Document. IEEE Std 1362-1998*, IEEE Computer Society Press, 1998.
2. Department of Defense, Topic N991-079 Multi-Level Security for Computer-Intensive Environments, *1999 DoD SBIR/STTR Program Solicitation*, U.S. Government Printing Office, 1999.
3. Forsberg, K. and H. Mooz. *System Engineering Overview. Software Requirements Engineering.* 1996. IEEE Computer Society Press.
4. Gomaa, H. The Impact of Prototyping on Software System Engineering. In *System and Software Requirements Engineering.* R. Thayer and M. Dofrman, eds. IEEE Computer Society Press. 1990.
5. Boehm, B. A Spiral Model of Software Development and Enhancement, in Thayer ed. *Tutorial: Software Engineering Project Management* IEEE Computer Society Press, 1988.
6. Davis, A. E. Bersoff, and E. Comer. A Strategy for Comparing Alternative Software Development Life Cycle Models." *IEEE Transactions on Software Engineering*. IEEE Computer Society Press. 1988.
7. Fairley, R. and R. Thayer, The Concept of Operations: The Bridge from Operational Requirements to Technical Specification, *Software Engineering*, M. Dorfman, and R. Thayer, eds. IEEE Computer Society Press. 1996.
8. Groguen, J. and C. Linde. Techniques for Requirements Elicitation. *Proceedings of the International Symposium on Requirements Engineering*. IEEE Press, 1993.
9. Ford, W. *Computer Communications Security Principles, Standards, Protocols, and Techniques*, PTR Prentice Hall, 1994
10. Pfleeger, C. *Security in Computing*, PTR Prentice Hall, 1996.

About the Author



Darwin E. Ammala is a senior-level software engineer with MPI Software Technology Inc., which has performed contract work for the National Science Foundation, Navy, Department of Energy, and NASA JPL. Previously, he was a senior computer scientist with 14 years of experience at Fort Mead, Md. He is pursing a master's degree in science from Mississippi State University, with an emphasis in security in high-performance and distributed-cluster computing.

MPI Software Technology Inc.
101 South Lafayette ST.
Starkville, Miss. 39759
Voice: 662-320-4300, ext. 11
Fax: 662-320-4301
E-mail: dammala@mpi-softtech.com



Software Engineering Education: On the Right Track

By John W. McCormick
University of Northern Iowa

An introductory undergraduate course in real-time embedded software development should acquaint students with the fundamental scientific issues of real-time computing and practical skills in software development. While the theoretical issues can be covered without a laboratory, real-time embedded programming skills require the experiences that a laboratory provides. A major problem is finding equipment suitable for teaching these skills.

Nearly all modern devices contain embedded software. Even a few years ago, the typical new car from General Motors contained $675 of steel and nearly $2,500 of electronics, including a dozen or so embedded microprocessors. Often, the software in these embedded systems must execute in real-time for the equipment to function correctly. Despite the need for skilled real-time embedded software developers, there is little attention paid to this area of software development in the undergraduate computer science curriculum.

An introductory undergraduate course in real-time embedded software development should acquaint students with the fundamental scientific issues of real-time computing and practical skills in software development. While the theoretical issues can be covered without a laboratory, real-time embedded programming skills require the experiences that a laboratory provides. A major problem is finding equipment suitable for teaching these skills.

Simulators are commonly used to give students experience with real-time programming. Typically these simulators do not provide many of the frustrating problems associated with physical systems. Hardware and software development are parallel activities in many embedded systems projects. Gathering evidence for the determination of whether a fault is in the hardware or the software is an important skill for the embedded systems programmer. Lack of experience with real systems is one reason cited by engineers who would exclude computer science graduates from their development teams.

For more than a decade I have used a computer controlled model railroad in my real-time embedded systems course. Some advantages of using a model railroad in the laboratory are that:

- Model railroad equipment is readily available and priced well below typical laboratory equipment.
- Model railroads provide a wealth of problems from both the discrete and continuous real-time domains.
- The electronics are easily understood by most undergraduate computer science students.
- Students are highly enthusiastic about writing software to control a model train layout.

As a direct result of presentation and publication of previous work [1], [2], [3], [4], more than 50 organizations have requested detailed specifications of the laboratory. All but three were discouraged by the amount of effort (500-plus hours) required to assemble the necessary interface electronics. With the support of the Maytag and Rockwell Foundations, I am implementing an *affordable* real-time embedded systems laboratory that other institutions can easily duplicate.

The Real-Time Systems Course

The computer science curriculum at the State University of New York (SUNY) at Plattsburgh includes a specialized track, Computer Controlled Systems. This track was developed for students interested in the specification, design, and implementation of real-time embedded software. In addition to the typical courses in a computer science curriculum, this track includes more courses in continuous mathematics, physics, and electronics. The departments of computer science and industrial technology at the University of Northern Iowa (UNI) are designing a joint computer controlled systems curriculum.

The real-time systems course serves as the curriculum's capstone course. To perform well in this course, students must integrate knowledge from their previous work in computer science, electronics, English, mathematics, and physics. Students are exposed to the fundamental scientific issues in real-time computing and gain practical skills of software development. A major goal is to train software engineers capable of working as members of an interdisciplinary development team. Many topics are covered at a survey level. For example, students in the course learn just enough of the basic concepts of control theory to be able to communicate with a control engineer and to implement a simple control algorithm. Feedback from employers in a wide range of domains, including avionics, communications, manufacturing, and medical instrumentation has been extraordinarily positive.

Laboratory Assignments

The four credit-hour course has three 50-minute lectures and a three-hour laboratory session each week. The early laboratory sessions are used to review (or learn) and practice with the features of the implementation language that are important for the completion of their project. These include data modeling, encapsulation and reuse, concurrent programming, and exceptions. Later laboratory sessions are devoted to developing code that will be directly applied to their projects, including polling and interrupt-based device drivers, implementation of a whistle class, and implementation of a turnout class

Turnouts are electromechanical devices that sometimes fail to operate correctly. The software must detect and correct turnout failures. Students derive their code from state machines they develop in one of the lecture sessions.

Course Project

Students are divided into teams of three or four students to complete a substantial (12K–15K lines) project. Teams are free to

formulate their own projects. Minimum project requirements are:

- Running multiple trains.
- Having at least one train controlled by a human engineer.
- Experiencing no collisions.
- Detecting and recovering from hardware failures, such as turnouts, sensors, lost cars, and devious professors.

Over the years, train races, train wars, and scheduling problems have been the most popular project themes. Deliverables for the project have included:

- A system concept document.
- A detailed user's manual.
- Object Modeling Template documents.
  - Object model diagrams.
  - Dynamic model diagrams.
  - Functional model diagrams.
  - Data dictionary.
- Compiled class specifications.
- Unit (class) test plans.

These deliverables are used as milestones throughout the course to help ensure that students keep up with the demanding schedule necessary to complete the project. One of my major tasks is to work with teams on their systems concept document to reduce overly optimistic proposals into ones that can be completed. Students are aware of the completion rates of past teams (presented later in this paper) so they understand that they can complete the project by the end of the semester.

Student teams do exhaustive module testing where behavior of a particular object (a turnout or locomotive) is well understood. Integration testing is bounded by the end of the semester.

The Laboratory

My first model railroad laboratory was constructed in 1983 at SUNY Plattsburgh. Construction of a new railroad layout at the University of Northern Iowa is under way.

Railroad Hardware

The model railroads are HO scale.[1] While smaller scales would permit more equipment in the laboratory, they are more expensive, more difficult to maintain, and less readily available.

To run multiple trains on their layouts, model railroaders traditionally divide the track into electrically isolated sections called blocks. Many toggle and rotary switches are used to connect a particular power supply (called a cab) to a group of track blocks beneath each train. In our layout, the computer controls the voltage and polarity applied to each of the blocks. Our current UNI layout design has 40 blocks. Today's model railroad enthusiasts often use more modern direct digital control of locomotives to solve the problem of multiple train control. We have rejected this approach as it provides fewer software development problems to our students and less experience with analog electronics.

Turnouts are controlled by gear- and screw-driven switch machines. The computer can determine and modify the state of each turnout. Our current UNI layout design has 26 turnouts.

In order to do closed loop control, it is necessary to obtain feedback on the process being controlled. For the model train this feedback consists of the trains' locations as a function of time. This information is obtained from:

- Hall effect sensors installed on the track. These are triggered by small magnets attached to the front of every locomotive and to the rear of each caboose.
- A radio link installed in a box car that sends a pulse with every wheel rotation.

Our UNI layout design has 55 Hall effect sensors. The radio link allows us to determine a train's position information to within about 1 centimeter. From the data obtained from the link we can also calculate the train's speed. Currently there are two problems associated with the radio link. The wheels on the boxcar slip on the track as the car moves, thus the calculated distance moved by the train is less than the actual distance. This error increases with time. This problem is a *good* problem as students can easily correct for the slippage by using the positions obtained from the fixed sensors. The second problem is a result of recycling radio transmitters from very inexpensive toys. The transmitters broadcast over a large portion of the frequency spectrum, making it impossible to use multiple transmitters at the same time. We are working on a design to replace the radios with an infra-red link.

A final piece of railroad hardware is a hand-held control cab. This is a small box with buttons, knobs, and toggle switches that a human engineer can use to control a train. Typical student projects assign knobs for train throttles, buttons for whistles and brakes, and toggle switches for train direction (forward or reverse) and for setting the next turnout ahead of the train (left or right).

Computing Hardware

A number of different hardware configurations have been used over the long history of this project. In our first laboratory, students developed their control software on a Digital Equipment Corporation PDP 11/24. They used a serial link to download executable programs to a PDP 11/23 computer. In 1989 I received a laboratory improvement grant from the National Science Foundation (NSF) enabling me to replace the 11/24 with a microVAX II and the 11/23 with an rtVAX (optimized for real-time). The system now under design at UNI uses PCs for software development. Two or three inexpensive networked microcomputers will boot and execute the software students developed.

Interface Hardware

The interface hardware connects the control computers to the railroad hardware. Figure 1 is a diagram showing the layers in the system. One or more CPUs are connected to commercially available analog-to-digital converters (ADC), digital-to-analog converters (DAC), TTL level digital I/O (DIO), and counter/timers. The connection may be made through any of a number of different buses such as ISA, EISA, PCI, GPIB, CAN, USB, and even standard serial or parallel ports. We use custom hardware to connect these devices to the railroad layout. In the past, this interface layer was handbuilt on wire wrapped and soldered prototyping boards. It took considerable effort to construct it. With the support of the Maytag Foundation and Rockwell, we are designing and manufacturing circuit boards that will make this aspect of building the laboratory much easier for us and other schools that wish to duplicate our efforts. The interface hardware consists of three subsystems (block control, turnout control, and train sensors) detailed in the following sections.

CPUs	Buses	Devices	Maytag Rockwell Custom Interface Boards	Railroad Layout
	ISA PCI GPIB CAN USB, etc.	DAC ADC DIO Counter/ Timer		Blocks Turnouts Sensors Cabs

Figure 1. *Hardware layers connecting the control computers to the model railroad*

Block Control

The block control subsystem controls the voltage and polarity applied to each track block in the railroad layout. Figure 2 shows a single track block circuit.

The two analog outputs are connected to the rails of a track block to supply power to the train on that block. Each circuit has four digital inputs and eight analog inputs. Three of the digital inputs (cab select) are used to select which of the eight analog inputs will be used to power the track block. The remaining digital input is used to select the polarity of the voltage applied to the track. The analog inputs (*cab voltages* in Figure 2) may be supplied by digital-to-analog converters or by programmable counter/timers. The latter uses pulse width modulation to control the speed of a train. We expect that each of our block control boards will contain six- or 12-block control circuits.

Turnout Control

This circuit controls a Tortoise brand switch machine. These switch machines take three to five seconds to change the direction of a turnout. There are four possible states for a turnout: left, right, moving left, and moving right. Each circuit uses one output bit to set the direction of the turnout. Rather than use two input bits to determine the state of the switch machine we use the output bit in combination with one input bit that reports whether the turnout has reached the desired direction.

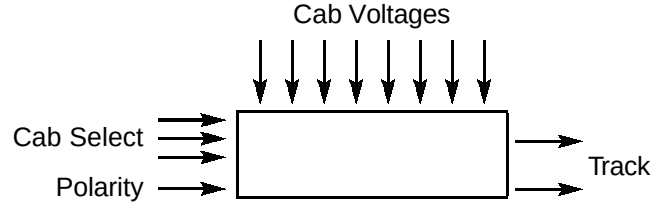
Train Sensors

This circuit connects the Hall effect sensors on the track to a DIO board with interrupt capabilities. We place these sensors on the boundaries between track blocks. When a locomotive is detected, the software must power up the next block before the wheels bridge the gap between blocks. This is a hard real-time deadline in the system as the block power supply fuse will blow if the software fails to power the next block in time .

Software

During the first six years that the real-time systems course was offered, students developed their control code in C. As shown in Figure 3, no team successfully implemented minimum project requirements when the C language was used. To ease student and teacher frustrations I made an increasing amount of my solutions available to the teams. Figure 3 shows that even when I provided nearly 60 percent of the project code, no team was successful in implementing the minimum requirements.

Figure 2. *Track Block Control Circuit*



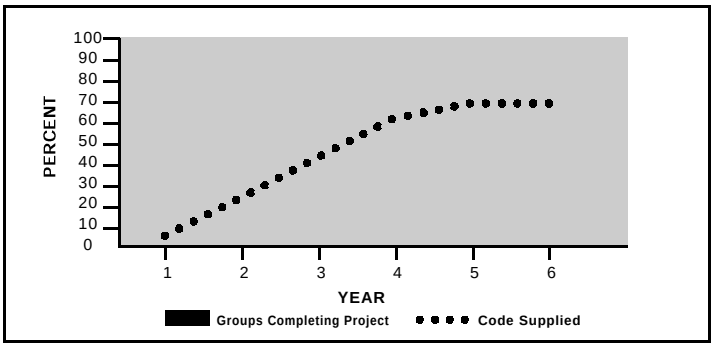


Figure 3. *C Language: Completion Rate (zero) and Amount of Code Supplied*

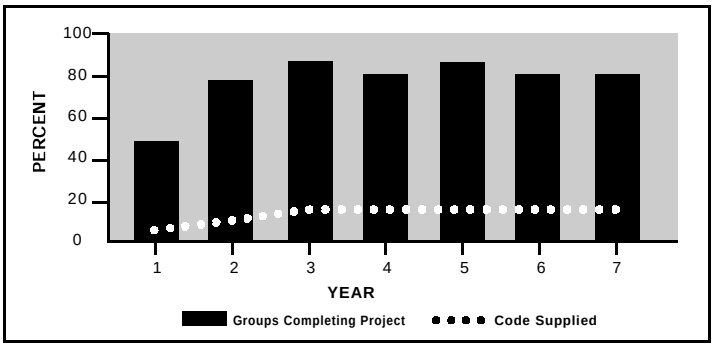
Along with the new hardware provided by the NSF funding was a collection of DEC compilers. Thinking that the low level of tasking provided through semaphores was the major contributor to the problem, I selected a language with a much higher level of tasking abstractions—Ada. I expected a disaster the first year with the new equipment and new language. As in a real-life embedded systems project, I was building the hardware while my students were writing the software. I finished the hardware with only four weeks remaining in the semester. But to my amazement, nearly 50 percent of the student teams had their projects working before the end of the semester. I had only supplied them with two sample device drivers. As shown in Figure 4, when I supplied some additional software components (simple window packages not relevant to the real-time aspect of the project), more than 75 percent of my teams routinely completed their projects.

Why Ada succeeds where C fails.

The only difference between the years in which teams succeeded in implementing their projects and those in which no team succeeded was the implementation language. The project specification, design, and unit testing techniques did not vary. While the new computing hardware the Ada teams used was more modern (faster and fewer breakdowns), it provided no significant implementation advantages. Upon reading the project listings and team member diaries, I concluded that the major advantages of Ada for these students were, in order of importance:

- Modeling of scalar objects.
  - – Strong typing.
  - – Range constraints.
  - – Enumeration types.
- Parameter modes that reflect the problem rather than the mechanism.
- Named parameter association.
- Arrays whose indices do not have to begin at zero.

Figure 4. *Ada Language: Completion Rate and Amount of Code Supplied*



- Representation clauses for device registers (record field selection rather than bit masks).
- Higher level of abstraction for tasking (rendezvous rather than semaphores).
- Exception handling.
- A compilation model that detects obsolete units.

I found my original hypothesis, that the major problem was C's low-level tasking mechanism, to be incorrect. While Ada's high level of abstraction for tasking was helpful to the students, it was the accurate modeling of scalar quantities that contributed the most to Ada's success in this course. This is consistent with studies done on the nature of wicked bugs in software [5] where nearly 80 percent of programming errors in the C/C++ programs studied were a result of problems with scalars.

Conclusions

The model railroad provides an exciting environment for teaching a course in real-time embedded systems. With the support of the Maytag Foundation and Rockwell, we are developing the interface hardware to allow us and other schools to easily connect a variety of computers to a model railroad at minimal cost. UNI will make the interface boards we design and manufacture available to all. Contact me for details.◆

References

1. McCormick, J.W. (1988). Using a Model Railroad to Teach Digital Process Control. *SIGCSE Bulletin*, 20, 304-308.
2. McCormick, J.W. (1991). A Laboratory for Teaching the Development of Real-Time Software Systems. *SIGCSE Bulletin*, 23, 260-264.
3. McCormick, J.W. (1992). A Model Railroad for Ada and Software Engineering. *Communications of the ACM*, 35, 68-70.
4. McCormick, J.W., Kudrle, J., & Poulin, J.M. (1994) Ada, Objects, and Model Trains. *The Proceedings of the Eighth Annual Software Engineering Education and Training Symposium*, Albuquerque, N.M., 8, 29-33.
5. Eisenstadt, M. (1997). My Hairiest Bug War Stories. *Communications of the ACM*, 40, 30-37.

Note

1. HO scale [half + O (gauge)] is a scale of 3.5 millimeters to 1 foot used especially for model toys (as automobiles or trains).

About the Author

John McCormick is professor and head of the computer science department at the University of Northern Iowa. Previously, he was professor of computer science at the State University of New York at Plattsburgh, where he received the Chancellor's Award for Excellence in Teaching. He is the author of two Ada-based textbooks for introductory computer science courses. He received his bachelor's degree from Pennsylvania State University and his doctorate from the University of California at Los Angeles.

University of Northern Iowa, Computer Science Department
Cedar Falls, Iowa 50614-0507
Voice: 319-273-2618
Fax: 319-273-7123
E-mail: mccormick@cs.uni.edu

Coming Events

August 6-11
6th Annual International Conference on Mobile Computing and Networking
www.research.telcordia.com/mobicom2000

August 7-8
IEEE Workshop on Memory Technology Design and Testing
http://pcgipseca.cee.hw.ac.uk/cec2000

August 17-19
Designing Interactive Systems (DIS)

September 10-12
Collaborative Virtual Environments (CVE)

September 10-14
Very Large Databases (VLD)
www.acm.org/events has information on DIS, CVE, and VLD.

September 18-19
The Internet Challenge—The Utility Response to a .Com World
www.tdworld.com/marketing/interchall.htm

September 26-28
2nd Computer Security & Information Assurance Conference
www.certconf.org

October 15-19
Object Oriented Programming Systems Languages and Applications Conference (OOPSLA 2000)
www.acm.org/events

October 23-25
4th Symposium on Operating Systems Design and Implementation
www.usenix.org/events/osdi2000

October 30-31
3rd International Conference on Practical Aspects of Knowledge Management (PAKM 2000)
www.do.isst.fhg.de/workflow/events/index_e.html

November 10
Infomation Outlook 2000 (Australian Computer Society)
www.acs.org.au/act/events/io2000/index.html

November 16-17
ACM Conference on Universal Usability
www.acm.org/sigchi/cuu

December 4-7
International Conference on Power System Technology
www.ee.uwa.edu.au/~aips/powercon

December 11-13
Global Development Network Conference
www.gdnet.org

April 29-May 3, 2001
Software Technology Conference 2001
www.stc-online.org

Don't Say the 'P' Word

By Lori Pajerek
Lockheed Martin Federal Systems

Process maturity has been extensively analyzed and codified, and the goal of process maturity has become pervasive throughout industry, government, and academia. The Software Engineering Institute estimates that there are more than 30 process maturity models and more are being developed. 'Process' has been a buzzword for a long time, and it may seem at times that more attention is paid to the process used to produce a product than to the product itself. This article examines the background experiences that led to developing process maturity models, deconstructs some of the arguments that have been posited—both for and against—and discusses some lessons learned.

The problems linked to software development have retained their challenge since they were first documented by Frederick Brooks in the *Mythical Man-Month* [1]. Then as now, software-intensive development projects have been plagued by lack of predictability, schedule and cost growth, failure to meet requirements, and as a direct result of these, low customer satisfaction. We see numbers quoted that show how dismal the state of the art is; that the vast majority of software projects are failures [2]. We have searched for solutions to these problems, seemingly ever since the days of Ada Lovelace.

There has been no lack of proposed solutions. Alan Davis chronicled a list of fads that have appeared on the software scene every few years since the 1970s [3]. Structured programming, object orientation, reuse, commercial off-the-shelf (COTS) products and others have had their day in the sun. Each has been heralded as a silver bullet, which none has been. This is not to say these ideas have no value; rather, that one must separate the substance from the hype. We incorporate what is valuable and make it standard practice. Each is recognized as one piece of the puzzle, not the whole solution. Progress is achieved slowly, small gains are made with each step. Because the problems remain largely unsolved, and because each of these doctrines is introduced with such fanfare, we are repeatedly seduced by the promise of a Holy Grail. Davis categorizes process maturity as one of these fads.

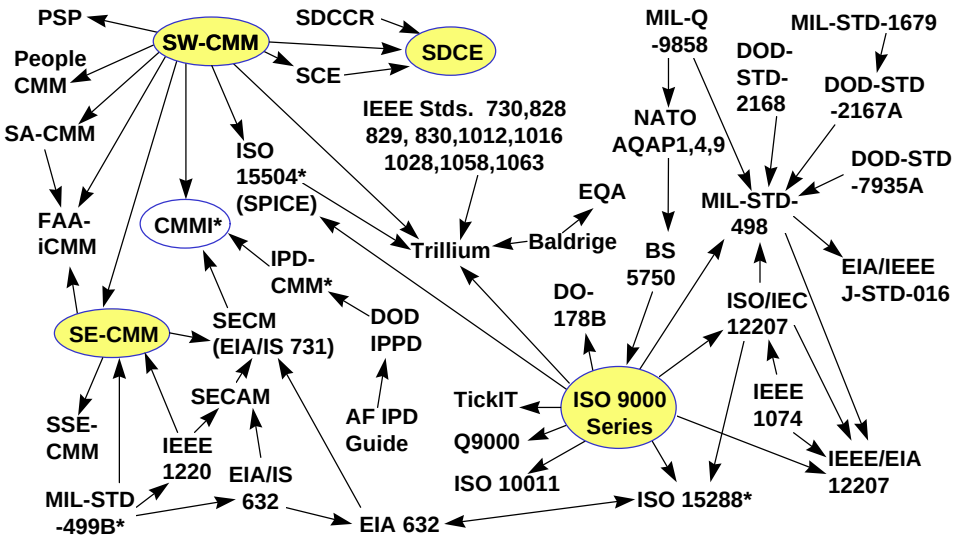
When Sarah Sheard writes about the Frameworks Quagmire [4], she is reflecting some of the frustration of those attempting to comply with a confusing and sometimes conflicting set of process dictates. One look at her pictorial representation (see Figure 1) is enough to make us say that this has gone too far. The word 'quagmire' evokes the feeling by many that they are drowning in too much process. Other writers note that there is a growing body of opinion that the practices that the Software Engingeering Institute's Capability Maturity Model (SEI CMM®) advocate are justified only for large and complex projects [5]. I can hear the debate now. Let's listen:

The Debate

Devil's Advocate: Process champions are quick to cite data to support the contention that organizations that adopt better processes produce better software. However, this is far from proving a cause-and-effect relationship between the two. The data put forth as evidence would never hold up against the standards applied to hard science. Statisticians know that there is a big difference between demonstrating a statistical coincidence and proving cause and effect. Because two phenomena coincide in a statistically significant sample does not necessarily mean that one causes the other.

Statistical coincidence is never accepted as proof of cause and effect. There must be additional empirical evidence to prove that one thing is the direct cause of another. Organizations that have good processes and also produce good software may be a coincidence, and may be due to a third factor, such as exceptionally good software engineers. Davis makes the point that with the right people you can succeed without process maturity, but that the best process in the world will not make you successful if you have the wrong people. It seems likely that if an organization is blessed with good engineers, it will be able to create good processes as well as good products. Or perhaps it is because companies that

Figure 1. *The Frameworks Quagmire*



* Not yet released

make good software also make money—enough money to support the overhead of process management.

The Rebuttal

Process Advocate: Unfortunately, we probably can not expect ever to have the kind of data that would satisfy a scientist. How could you ever really trace the *goodness* of a certain piece of software back to first causes with any certainty? You can only observe the statistical coincidences and draw inferences. Is that not good enough? If the two things go hand-in-hand, is it unreasonable to think that obtaining one of the two will increase your chances of obtaining the other? Remember Jay Forrester's warning that "intuitive judgments about cause-and-effect relationships may not be effective in complex feedback systems . . . with their multiple feedback loops and levels. Complex systems have a multitude of interactions, not simply cause-and-effect relationships. Causes may not be proximate in time and space to effects [6]." Certainly this comment applies to the complex relationships inherent in a development process.

Good software may be developed without a good process—*once*. That is an accident. Good processes help ensure that good software can happen more than once. A smart hacker can write a killer application, and undoubtedly makes some mistakes along the way. To an individual, those mistakes may not matter very much. Time and effort wasted are probably of little consequence. But in an organizational environment, mistakes *do* matter, because they consume valuable resources. Nobody ever sees software scrap, but it affects the bottom line as surely as hardware scrap. Can you imagine any manufacturing manager tolerating the amount of scrap coming off his production line as is routinely accepted in our software factories? Software scrap may not represent an expense in raw materials, but costs are incurred in terms of labor and schedule time, both of which are usually in short supply.

Devil's Advocate: I will acknowledge that software scrap is bad, but what makes you sure that process maturity will reduce it? Many companies are investing large amounts of money to raise their CMM maturity level, but do they really know if they will get the expected payoff? They will at least get bragging rights to a high SEI rating, which will probably be an indirect cause of increased revenue. But it may be that they would get a better value by using that money to hire the best engineers they can. It is hard to say, because we can not be sure which is the cause and which is the effect.

Process is merely a means to an end. It actually is a combination of ends: technical, quality, cost, and schedule performance, that will ultimately lead to customer satisfaction. There is evidence that companies with mature processes do indeed achieve these objectives. But even accepting the inference of a cause-and-effect relationship between the two, you must still consider whether there might have been another way to meet that goal. This is not to suggest that process maturity is bad, but that you should think about why each process step exists, and whether there is a better (i.e., easier or more cost-effective) way to achieve the same end. Furthermore, organizations can not afford to lose sight of the fact that achieving the highest levels of quality and customer satisfaction may not be the pre-eminent goals. They must also make enough profit to stay in business.

Process Advocate: You are right, process exists as a means to an end. Individual process steps exist for a variety of reasons —technical, cost, schedule, legal. Some steps are recommended as best engineering practices; they are there to promote technical quality. Other steps within the same process are there only for management reasons. If we pretend that we have infinite time and money to complete a project, we would not have to employ a lot of management controls. Since that situation never occurs, we have to find ways to optimize technical quality while maintaining some schedule and cost limits. Because these controls have to be embedded in the engineering tasks, our engineering processes contain a mixture of management and technical procedures. The advent of initiatives like Integrated Product Development have intertwined them even more, making it harder to separate the strands.

Software has given us the capability to build systems that are far more complicated than before its advent, and they are becoming more complicated all the time. Now that we are building systems containing millions of lines of code, development must be parceled out. Process is what makes this division of labor succeed; just being smart does not carry it off any more. Documented processes become the repository for an organization's collective wisdom and experience—lessons learned about what works and what does not.

Devil's Advocate: You agree, then, that less process is needed for small programs. Many organizations recognize this fact intuitively, and let smaller programs off the hook. That helps relieve the burden, but it does not help them figure out what *is* appropriate for programs their size. Even for large programs, one of the process mantras is that "the process must be tailored for your program." Practical tailoring guidance is hard to come by and difficult to apply.

Process Advocate: Yes, many of the sub-processes of systems engineering are really systems management. They exist, like the science of systems engineering, because of the need to manage large, complex efforts that involve many components, interfaces, people, sites, companies, etc. They are needed to keep a large project organization moving in sync towards a single goal at the right time. These subprocesses are more easily waived when you have a small team and a short schedule.

One mental exercise that can help is to assess each process step against the following questions:

- Would I do this if I were building this product by myself?
- Why is this process step here; is it for a technical or a managerial reason?

This will help to identify and segregate the steps that exist primarily to serve management goals. Those may be negotiable on small projects. For example, process steps that require completion of checklists, preparation of status reports, or multiple levels of approval to do things can probably be simplified or eliminated.

People fear that processes are going to tell them how to work. The most important processes for an organization to mature do not prescribe how to work but how to coordinate. No one minds being

told how to get others to do what they want them to do. That is the function of most good processes. A newly minted engineer may not initially understand why he has to follow the process—he never had to do this in college. Eventually he sees the value when he realizes he is no longer working by himself, and his success is dependent on what others do.

When Bad Things Happen to Good Engineers

It must be understood that process maturity can not be realized by the efforts of engineers alone. As Watts Humphrey has noted, poor project management will defeat good engineering, and is the most frequent cause of project failure [7]. When managers insist that engineers shortcut the best engineering practice due to schedule or budget pressures, process maturity fails. These managers are often responding to inflexible contract demands to which their company committed in order to win a competitive bid. The procurement process encourages bidders to submit proposals that set unrealistic schedule and cost targets. Hence, while the customer community may be professing the desirability of a contractor's high level of process maturity, the incentive to industry promotes the exact opposite result. Thus, another prerequisite for a mature development process to thrive is the co-existence of a mature procurement process on the customer's part [8]. Even then, reality has a way of subverting the best intentions. Although everybody wants to accrue the benefits of capable processes, managers often experience 'sticker shock' that causes them to cut corners.

A hallmark of a mature development process is emphasis on early requirements analysis and up-front planning. This requires program schedules and budgets to be more heavily loaded on the front end. Despite having heard the caution, "Pay me now or pay me later," some program managers think they can get away with not paying at all. This includes managers in both customer and contractor organizations. Budgets and schedules are drawn up optimistically, trusting in a best-case scenario. This is rarely the scenario that unfolds, and the fact that it does not is largely a self-fulfilling prophecy.

Spending a lot of time and money up front is an expression of faith. It manifests the belief that heavy investment in thorough requirements analysis, trade studies, etc. will prevent problems later, and that the cost of fixing those problems would be greater than the initial investment. But it is difficult to quantify the cost of problems that never happen, so it is hard for program managers to commit to spending lots of resources early, when the payoff is so intangible. Despite numerous studies to support its validity, few programs are scheduled and budgeted this way. There is never sufficient time and money to do all the tasks that engineering best practice would dictate, i.e., to follow the process! Projects seem to be programmed for failure before they even start. Once the inevitable problems occur, projects revert to crisis management mode, which is not noted for its high level of process maturity.

While engineers working in the real world are told to follow best practices, circumstances often make it difficult for them to do so, and they are blamed when they fail. Is it any wonder they become annoyed and think that the Process Police issue lofty dictates from the ivory tower, while they struggle in the trenches to get the product out the door? Who can be surprised that engineers fighting daily fires do not want to hear the 'P' word? They want to do the right thing, but it seems as if their hands are tied when, for example, they have six months worth of requirements analysis activity to squeeze into the four weeks allotted in the schedule. Watts Humphrey also states that if engineers can reasonably defend their plans, management should respect these plans and not override them with schedule edicts. Too often, this caution is ignored.

Where Process Models Fall Short

Investing time and money up front should not require a leap of faith. We ought to be able to draw up a front-loaded schedule and budget with a reasonable degree of certainty that we can shorten the back end without undue risk. We can not do this today because of deficiencies in our process models and our base of collected measurement data. The reason we create models is to validate designs by varying operational parameters and conditions, and observing the results. Engineers trust their models because they trust the data that goes into them. Program managers, however, can not place the same level of trust in process models because we lack the abundance of hard data to substantiate them.

Many current process models are deficient because they do not always view the processes as systems. The process is the system an organization uses to generate all its other systems, i.e., its products. However, few organizations apply the same engineering rigor to creating their development processes that they apply to the systems developed for external customers. A properly engineered development process would be modeled in such a way that managers could accurately foresee the outcomes of various hypothetical actions. What happens to Integration and Test (I&T) time if we cut the Requirements Analysis time in half? Who knows for sure? We can probably guess correctly that it will increase, but by how much? Ideally, we could vary these parameters in a formula to find the optimum balance.

An example discussed previously provides a further illustration of this point. We have all seen empirical data to support the premise that the cost of correcting system defects increases as the system development progresses into later phases. But process models have not fully instantiated this data to the point where we can quantitatively predict the downstream effects of qualitative changes to the process. In an ideal model, you would be able to calculate how much time devoted to requirement reviews or design inspections is required to remove a certain percentage of defects, and at what point the law of diminishing returns indicates that the I&T savings no longer offset the cost of additional up-front reviews. The accuracy of such a model depends on having a substantial amount of validated measurement data, something that few organizations possess today. It is essential that we collect this data if we are to arrive at the point where our process models can be sufficiently calibrated to perform these sophisticated *what if* analyses.

Integrated Product Development notwithstanding, few process models in use today are truly integrated. Even with the best intentions, organizations tend to quickly decouple constituent subprocesses as they descend through a top-down

decomposition of their development process. Except for time sequence dependencies, the system-wide effects of changing one subprocess are unknown. Practitioners still operate within their traditional stovepipe processes, even though their command media and their management structure may proclaim them integrated.

Another shortcoming in current process models is that the rationale for each process step is rarely captured. If it were, this type of data could be of great value when attempting to tailor or reengineer a process. People are often afraid to change or eliminate a process step if they do not know why it is there. Worse, more aggressive individuals may rashly eliminate steps because they do not immediately see their value. If they were aware of the possible consequences, they could at least take the risk knowingly. While some organizations collect historical lessons learned information, it is often poorly organized, difficult to access, and not kept up-to-date. Most signficantly, it is not tied directly to the process model. A good process model would have links to these lessons learned to show when a process step was changed, added, or deleted as the result of a lesson learned. We often maintain rationale for other engineering decisions, why not for processes? Capturing rationale for technical decisions is another one of the tenets of a mature engineering process. This is just another example of how we need to engineer the development process with the same rigor we engineer other systems.

Taking the Long View

Continuous Process Improvement (CPI) sounds like a good thing, and it is. The point is not that CPI is an impossible or unworthy goal, but that like the process, CPI is a means to an end. There may well be alternative routes to the same end. Organizations pursue CPI because they believe it will increase productivity and quality while reducing the cost of doing business. However, even the authors and champions of maturity models admit, when questioned, that there is no hard data to quantify the return on investment (ROI) in process maturity. Unless the goal is to achieve a CMM Level 5 rating for its own sake, it is valid to suggest that money spent on CPI may be better spent elsewhere in pursuit of the same goals. We should not forget that the law of diminishing returns will apply to CPI just as it does to any other investment. CPI requires a significant capital investment, with a promise of return on that investment. As with many corrective actions, the biggest ROI on CPI is achieved by a few heavy hitters. This is embodied in the well-established theory of Pareto analysis. For this reason, many practitioners believe that the difference between a Level 3 organization and a Level 1 organization is probably greater than the difference between a Level 5 organization and a Level 3 organization.

Once an organization has achieved a high level of process maturity, it is valid to question whether the ROI on continued improvement is sufficient to justify the investment. The gains will become smaller and smaller, and there may be a point where the investment is larger than the payback. Most organizations are still at relatively immature levels of process capability, and there are many valuable gains to be made. In this current state, it is hard to imagine that someday we will be at the point where all the low-hanging fruit has been picked. When that day arrives, we will have to take a hard look at the received value of pursuing CPI.

Also, there is a danger of confusing continuous improvement of the process with improving the process deployment. Deploying any reasonably adequate process rigorously and uniformly is of greater value than having a perfect process on paper, but not enforcing it effectively. An organization's priority, therefore, might be to ensure that a majority of programs within the organization are performing at Level 3 (for example), before investing in advancing to Level 5 for a limited number of programs.

The bottom line is that process maturity is only one of many factors that contribute to the ultimate success or failure of any project. There is no doubt that personal attributes such as education, training, and work ethic of the individuals executing the process will also have an effect. Likewise, the finest engineers can not perform up to their potential if not given an adequate working environment with sufficient resources of time, money, and tools. Finally, even the best engineers with the most ample resources may still fail if the project is badly managed in other ways. Process is just one ingredient in the mix.◆

References

1. Brooks, Frederick, *The Mythical Man-Month*, Addison-Wesley, 1975.
2. Alder, Rudy, Instead of the Wrecking Ball, *CROSSTALK*, May 1998.
3. Davis, Alan, Software Lemmingineering, *IEEE Software*, September 1993.
4. Sheard, Sarah, The Frameworks Quagmire, A Brief Look, *Proceedings of the Seventh Annual International Symposium of the International Council on Systems Engineering*, August 1997.
5. Hadden, Rita, How Scaleable Are CMM Key Practices?, *CROSSTALK*, April 1998.
6. Hughes, Thomas P., *Rescuing Prometheus*, Pantheon, 1998.
7. Humphrey, Watts S., Three Dimensions of Process Improvement Part I: Process Maturity, *CROSSTALK*, February 1998.
8. Courteney, H. and Ruston, S., Mature Procurement of Large Scale Systems: A Better Way to Buy, *Proceedings of the Seventh Annual International Symposium of the International Council on Systems*

Additional Reading

Gundrum, Valerie, Architecture for a Process Meta-System, *Proceedings of the Seventh Annual International Symposium of the International Council on Systems Engineering*, June 1999.

About the Author

Lori Pajerek is an Advisory Systems Engineer at Lockheed Martin Federal Systems in Owego, N.Y. Her current assignment in the Systems Engineering Technology department includes surveying, evaluating, selecting, and deploying Systems Engineering tools. With more than 15 years experience in systems and software engineering for defense-related industries, her areas of interest and expertise is requirements engineering and management. She has a bachelor of science degree in mathematical sciences from Binghamton University, and is a member of the International Council on Systems Engineering (INCOSE).

Lockheed Martin Federal Systems
1801 State Route 17C
Maildrop 0210
Owego, N.Y. 13827
Voice: 607-751-6226
Fax: 607-751-6025
E-mail: lori.pajerek@lmco.com

Don't Forget About Good Management

By Randall W. Jensen
Software Engineering Inc.

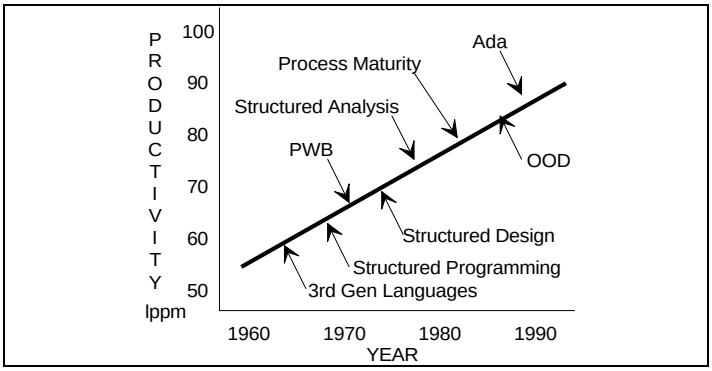
Dear Editor:

I must disagree with the premise of the December 1999 article *21st Century Engineer* on the grounds that good management, the most important productivity and quality driver, has been ignored. The concept of teams of good people using good tools within a good process (defined as gold collar workers) is commendable; however, the article's message implies that development technology alone is necessary for high performance teams. The following illustrates the point.

Defense industry software development productivity (average), measured from start of development through final qualification test, has grown almost linearly from 1960 through the present. A simplified (smoothed) productivity growth curve in the figure to the rightS shows this growth. The result, smoothed or not, shows a growth of software development productivity less than one source line per person-month per year over the entire 30-year period. During that time, each new technology has assured us the productivity problems of the past have been solved.

C. C. Tonies defined an effectiveness formula[1] that describes the net effect of an individual's effort in a software development environment. The effectiveness expression, $E = C[M(CS)]$, where E is the net individual effectiveness, C accounts for communication skills (0-1), M ranks management concept awareness (0-1), and CS ranks computer science technical ability (0-1).

Figure 1. *Average software development productivity growth 1960-1990*



B. W. Boehm suggested in *Software Engineering Economics* that poor management can increase software costs more rapidly than any other cost factor[2]. Boehm omitted management-related factors from the Constructive Cost Model (COCOMO) by assuming project management was uniform, constant, and good. In reality, management since 1970 has been uniform, not good, and unchanging.

G. M. Weinberg groups Boehm's cost impacts to illustrate the relative importance of each group. Figure 2 presents Weinberg's results emphasizing the importance of management in projecting software costs.

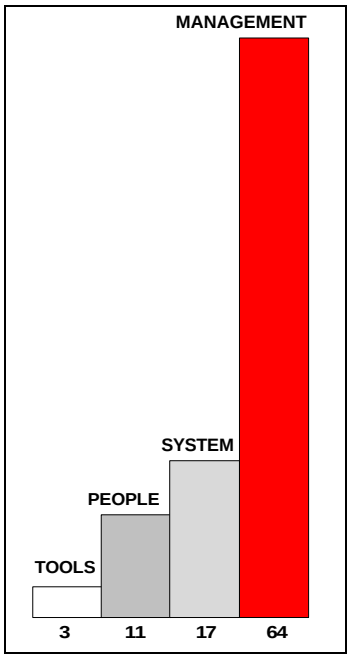
Contact Information

Dr. Randall W. Jensen, President
Software Engineering Inc.
660 North Highland Blvd.
Brigham City, Utah 84302
E-mail: seisage@aol.comz

References

1. Jensen, Randall W. and Tonies, Charles C. *Software Engineering*. Englewood Cliffs, N.J.: Prentice-Hall, 1979), pp. 8-9.
2. Boehm, Barry W. *Software Engineering Economics* (Englewood Cliffs, N.J.: Prentice-Hall, 1981), pp. 486-487.
3. Weinberg, Gerald M. *Quality Software Management, vol. 3: Congruent Action* (New York, N.Y.: Dorset House Publishing, 1994), pp. 15-16.

Figure 2. *Relative impact of COCOMO software cost driver groups*



GSAM Available on Software Technology Support Center Web Site

***The Guidelines for Successful Acquisition and Management of Software Intensive Systems (GSAM): Weapon Systems, Command and Control Systems, and Management Information Systems,* Version 3.0 May 2000, is available on the Software Technology Support Center's Web site at** www.stsc.hill.af.mil. **No hard copies will be available.**

The GSAM also will be included in the CD-ROM distributed by the Software Technology Conference 2000, that took place April 30-May 4 in Salt Lake City. Contact Vivian Johnson at Utah State University to request a copy. She can be reached at 435-797-0424 or vivianj@ext.usu.edu. **Interested persons can also obtain the *GSAM* in a future release of the *Defense Acquisition Deskbook.***

Call for Articles

Plan ahead to submit articles to be published in CROSSTALK in the next (we mean it this time) millenium.

January 2001	Modeling & Simulation
Deadline:	September 01, 2000
February 2001	Configuration Management
Deadline:	October 02, 2000
March 2001	Measures & Metrics
Deadline:	November 01, 2000
April 2001	Requirements Management
Deadline:	December 04, 2000
May 2001	Process Improvement
Deadline:	January 02, 2001
June 2001	Software Testing
Deadline:	February 01, 2001
July 2001	Software Quality
Deadline:	March 01, 2001
August 2001	DII-COE
Deadline:	April 02, 2001

A Tale of Two Monoliths

"Forcing Microsoft to include Netscape's competing software in our operating system is like requiring Coca-Cola to include three cans of Pepsi in every six-pack it sells." —Bill Gates, Microsoft

Like Microsoft, Coca-Cola has recently fallen on some *relatively* hard times. Not to shed a tear for either organization–both are still No. 1 in the world at what they do. But being the frontrunner increasingly has its drawbacks. Upon reaching the pinnacle the only way to go is down. Perhaps in today's world it is better to be a comfortable No. 2.

The results of the Microsoft antitrust litigation have been well publicized; Coke's travails over the past year or so, however, are less well known. In the words of stockbroker Roy Burry, "Coca-Cola, more than any other company, demonstrates what was so wonderful and now is not so wonderful." In the spirit of misery [or in this case, miserly] loves company, Coke losing its fizz is parallel to Microsoft being down to two bytes.

Coke's longtime CEO, Roberto Goizueta, retired. Perceived by many to be a miracle worker, he had increased Coke's market value from $4 billion to $145 billion during his reign. Even seeming disasters like New Coke were massaged into success by raising Coke Classic from its ashes. His successor, former Coca-Cola CFO Doug Ivester, cared more about numbers than about people. When contaminated Coke sickened consumers in Belgium in 1999, his apology was late and deemed insincere. African-American employees in Atlanta sued Coke for discrimination. Newly developed markets in Brazil and Russia failed to pay off because of poor market conditions. Chinese students and French café owners, angry at U.S. trade policy, boycotted efforts to purchase rival companies in Europe that were blocked for antitrust reasons.

Bill Gates retired as CEO, replaced by Steve Ballmer. Subsequently, the company lost both an antitrust suit and billions of dollars. Is there a connection?

Pepsi and new beverages such as SoBe carved into Coke's market share at home. Long a strength, Coke's ad campaigns seemed lackluster. The expensive practice of subsidizing retailers for shelf space was beginning to take its toll. Coke's relationship with Disney was jeopardized after cozying up to Universal, Disney's biggest rival. After a profit of $32 million on bottling in 1998, which was a 79 percent decrease from the 1997 level, Coke lost $184 million in 1999, the worst returns in more than 40 years. Ivester resigned.

Microsoft is dead! Long live Microsoft!

The new CEO and aptly named Douglas Daft immediately took steps to decentralize U.S. operations, downsizing 6,000 jobs (20 percent of the workforce). This still failed to get Coke out of its slump. The market still listed it a risk, and Standard & Poor's threatened to lower Coke's credit rating. Pepsi-Cola, meanwhile, was rated a buy. Mountain Dew, a software engineer staple, is a Pepsi product.

Wouldn't you like to own an iMac too?

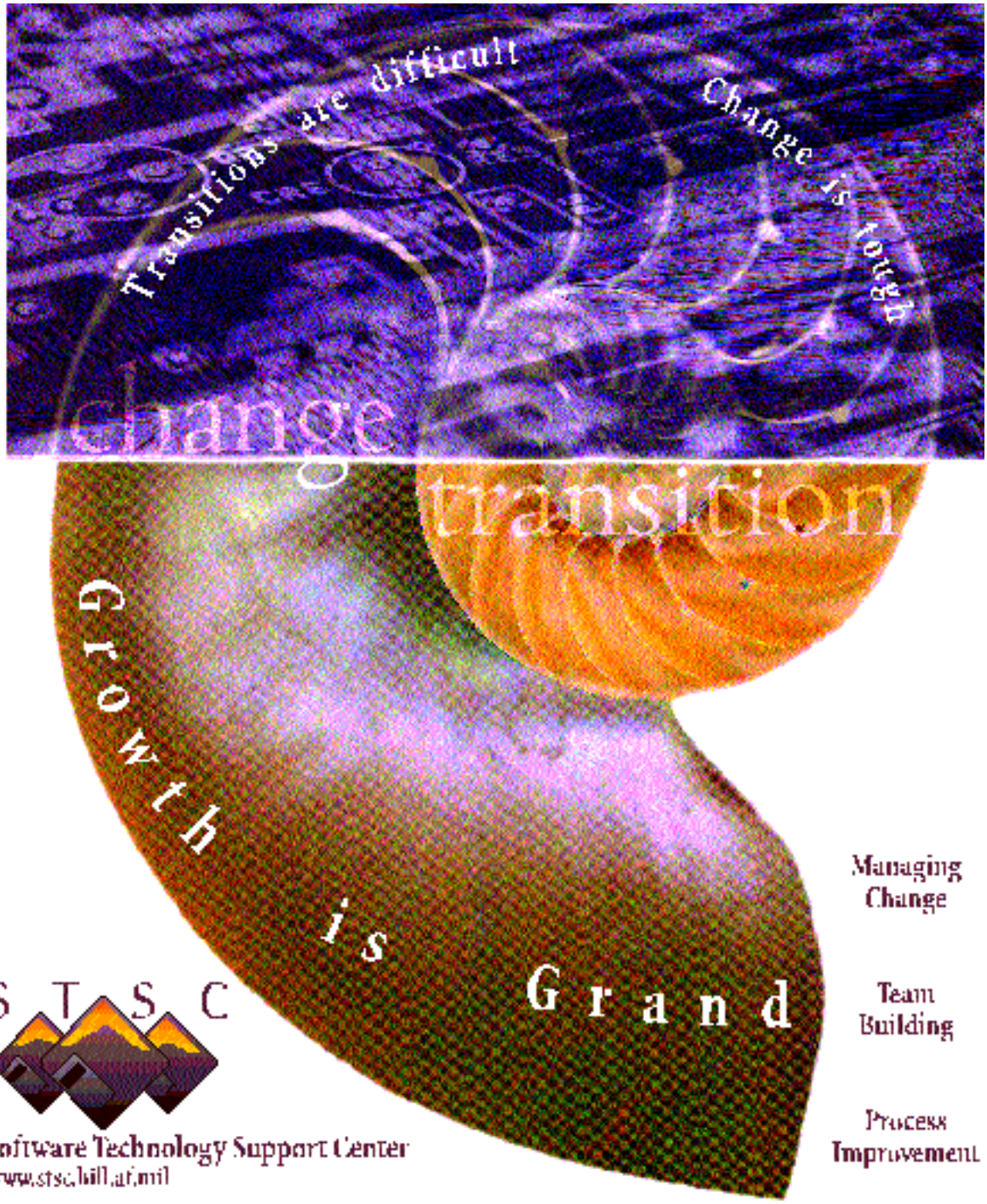
One new idea that has come from Coke is a machine that would automatically raise the price of beverages as the temperature climbs. Something tells me consumers will be steamed over this. Their full-scale launch of Dasani bottled water has only one problem-it tastes terrible. They recently gave their employees Friday afternoons off and another paid holiday (Coca-Cola's Birthday). That's what I call project management.

Which do you prefer—New Windows or Microsoft Classic?

Compared with Coke, Microsoft does not have it so bad. The Coca-Colans suffer because of something sweet and sparkling. Perhaps they should follow the path of John Sculley, formerly of Pepsi, whom Apple's Steve Jobs recruited with the question, "Do you want to spend the rest of your life selling sugar water, or do you want to come with me and change the world?"

The future again belongs to the savvy bridesmaids of the market. Or maybe not.

—Matt Welker



Managing Change

Team Building

Process Improvement

S T S C

Software Technology Support Center

www.stsc.hill.af.mil

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

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

BULK RATE
US POSTAGE PAID
Permit No. 481
Cedarburg, WI