Today's Software Complexity Demands Good CM

Reuel S. Alder, *Publisher*

To address the configuration management (CM) theme of this issue, I will first state unequivocally that documentation is good. A recent experience serves as a vivid demonstration of this point.

I just returned from a trip to Denver that was arranged entirely by electronic means. I had contacted my travel agents, given them my travel plans, and paid with my credit card. All I had to do was go to the airport. No paper was necessary—my ticket was electronic.

There was just one problem. As I entered the terminal, I thought I had stepped into a time warp. All the computers were down, as if the year 2000 problem had just struck. I could not get a boarding pass or seat assignment at the check-in counter. I was sent to the gate with the hope that the computer problems would be solved before I got there. I checked in at the gate, but since the computers were still down, I could not get a seat assignment.

When it was time to board the plane they followed a peculiar process: First, all passengers with seat assignments on their *paper* boarding passes were invited to board the plane. Then, all passengers with paper boarding passes and no seat assignment were invited to board. After numerous checks to see if any seats were left, those with electronic tickets were invited onto the plane; as we boarded, our names were written on a piece of lined paper to keep a record of who boarded the aircraft. I felt like a third-class citizen (not second class, because my ticket was coach). During this process, it was not clear to us electronic dependents if seating would ever be available.

The moral of this story might be that paper is superior to electrons, but that is not the point I want to make; instead, I intend to emphasize the necessity of keeping track through CM. CM is an important job, but apparently, few people want to do it. CM has moved beyond mere clerical work into a highly technical realm, but those with sufficient training to perform CM feel that the job is too mundane and monotonous, that it does not provide a creative outlet. They would prefer to engage in the more "glamorous" aspects of software production, such as design and code.

But CM cannot be neglected; it can mean the difference between chaos and order, between good software and defective software, between precision and error. The complexity of today's software demands that meticulous care be taken to track and record all modifications, deletions, and additions. No human mind is equipped to remember what was changed on dozens of versions.

But even if someone were able to retain that much information, they would not necessarily remain with the project until completion. When people leave organizations, their successors need to pick up where they left off with little delay. They need to know, by consulting the documents left behind, what was done and what was not. Duplication of effort and omitted tasks waste valuable time in projects that all too often are hurtling toward a deadline.

Just as the absence of written records nearly left me waving goodbye to my flight, poor CM in software projects could leave you bidding farewell to months of hard work as you spend an enormous amount of time backtracking, retesting, and recoding to discover why the modification you made did not turn up in the final product. This month's collection of articles will provide assistance to improve any CM effort. ◆

More Advice on CM Tool Acquisition

Ronald Starbuck's article, "Software Configuration Management: Don't Buy a Tool First" (*CROSSTALK*, November 1997), raised several interesting points. But there are three additional concerns that are key to finding the right software configuration management (CM) tool and for the successful implementation of a sound CM process that is valued for its benefits rather than despised for its perceived obstacles.

First, too often, organizations will purchase a CM tool based on its "bells and whistles" without examining whether those features are desirable and, more important, whether the tool rests on a sound CM process foundation. Second, the tool purchase decision makers may fail to view CM as a discipline that involves not only software engineers or developers but also program managers, test engineers, quality assurance specialists, and the customer(s). In so doing, they fail to evaluate a CM tool in light of the comprehensive needs of the organization and of those involved with or impacted by CM.

And third, organizations frequently find themselves purchasing a CM tool from a vendor who does not know CM. Organizations can reap crucial value-added CM expertise when they work with a vendor who has firsthand knowledge of CM and can apply that "lessons-learned" experience to an organization's unique culture and process, as indicated by Starbuck. Teaming with a CM tool vendor who knows and understands CM—the process and the people—can make the purchase of a CM tool one of the most crucial and ultimately successful steps an organization can take.

Debbie J. Stack
Progressive Software Solutions, Inc. (PROSOFT)

Stop-Gap Configuration Management

Ted Gill
Puget Sound Naval Shipyard

Configuration management can be highly difficult to implement at a site where the software development effort has been at the ad hoc level. Particularly for pre-existing development projects, to attempt to start a full-blown configuration management process is likely to be a lethal cure. This article outlines an exceptionally simple, goal-oriented "stop-gap" configuration management plan. The purpose of the plan is to gain immediate control of the most critical aspects of a development project and build a foundation of methods and understanding upon which a more complete configuration management plan can be built.

This article describes the most vital steps to bring essential aspects of the formal configuration management (CM) discipline to a project that has had none. This is emergency information—*triage*, pure and simple. If you use this information, the probable reason is that you are having a CM emergency. The typical condition in which a project finds itself at this point is something like this:

- The schedule is already extremely tight.
- The resources are barely adequate for the basic tasks, let alone "extras" like CM activities.
- No one on the project has extensive formal CM training or experience.
- The need for CM has just become apparent due to a big CM blunder on this or another project, a management CM mandate, or a milestone looming in the near future, and CM controls do not exist.

To team members in this situation, CM may appear big and scary. However, the essentials activities of CM are probably already being carried out in the project—albeit in an informal manner.

The suggestions and steps outlined here will in no way fulfill the requirements of a complete, coordinated CM plan. But they will go a long way toward establishing core processes and will mitigate the risks posed by a loss or lack of configuration control.

The following sections describe the important items and concepts for stop-gap configuration management.

Establish Version Control of the Code

Your primary goal must be to control the actual code: source, object, and executable. The following concepts will help you gain version control.

- Baseline the current code at some meaningful point in the development.
- Define what your computer software configuration items (CSCIs) are: individual modules, subsystems, etc.
- For each CSCI, determine its lowest level of decomposition, i.e., configuration units (CUs). This decomposition sets the granularity or resolution of configuration control.
- Determine an ironclad method to identify your CUs and their revision level. (Filename, date, and time will work if there is nothing else; if you have PVCS or a source code control system, use it.)
- Document your baseline with a version description document. This document should identify the baseline to which it refers (baselines need to be uniquely identified and catalogued), the CSCI it documents, and every CU in the CSCI and its revision level.
- Assign ownership of modules to specific developers—document the ownership and hold to it.

Establish Formal Control

Establish a formal method of change control for *all* changes to the code—accept no other method to propose changes. "Formal" means "with a form" and with a specific review and approval process. A change request form must include the following fields:

Submitter fields

- Project.
- Type (problem or enhancement). *(May include more detail.)*
- Severity.
- Date submitted.
- Description.
- Reporter or submitter.
- Unique identifying number.

Change control board fields

- Action (approve/disapprove/defer[until when]).
- Date acted on.
- Justification for disapprove/defer actions *(optional).*

Fields for those who act on the change request

- Date changed.
- CSCIs and CUs affected.

Additional Change Control Issues

- Always provide feedback to the submitter.
- Always document and save change requests, even if disapproved.
- When code is modified in response to change requests, developers should always document the change request number(s) on which they are working.
- The configuration control board (CCB) (see following section) must review and approve all changes to baselined CSCIs.

Establish a Project CCB

The CCB is responsible to approve all configuration changes to baselined configuration items. This ensures that all proposed changes receive a technical analysis and review and that they are documented for tracking and auditing purposes. The board also has final responsibility for release management, e.g., establishing new baselines.

The elements of the CCB already exist in a project that does any amount of formal change control. The goal must be to officially establish the board to gain formal control over the change approval process as it affects configuration control.

The basic tasks of the CCB are to declare baselines on CSCIs (promotions, releases, etc.), to review changes to baselined CSCIs, and approve, disapprove, or defer their implementation.

The above is a short but extremely important task list. The CCB must have a stranglehold on the project. Nothing can be changed without their approval—end of discussion. For this reason, board members must be chosen carefully. The board must be composed of representatives from all affected organizations or concerns (stakeholders) such as

- Functional or user community.
- Developers.
- Test group.
- Hardware design and operation personnel.
- Interface groups.
- Database administrators.

Although every member of the CCB may not be excited about every change, be certain that some change will affect every member of the board at some time. It should not be difficult to recall past experiences when an unwise or costly mistake could have been avoided if the right people had known about a proposed change.

The chair of the CCB must be from project management—a person who can unambiguously resolve conflicts within the board and enforce the board's decisions on the project. Decisions on change implementation and CSCI promotion translate directly to fundamental project cost, schedule, and quality issues. The CCB will find that their efforts are an infuriating exercise in futility if their decisions are continually reversed or ignored by an outside entity with the real decision-making authority. Do not let this happen; put that entity in charge of the board. By doing so, those who have decision authority are directly coupled to those who have expertise on the details. Decisions of the CCB should be reached by consensus whenever possible. The group dynamic must reflect the cooperative nature of a development project. The chair must nurture this cooperative vision and take unilateral action only when all other methods have been exhausted.

Document What You Do

As a treatise on triage, this article does not suggest elaborate plans. Beyond the specific documents required above, a written record is needed of what you are doing or plan to do. These plans do not need to be more formal than memorandums to all project team members—memorandums are adequate if they convey the plan and its execution.

There are two essential attributes of the documentation portion of stop-gap CM:

- The documents must describe what to do and how to do it.
- The documents must be published and disseminated so that all involved know the project status and what is expected of them.

Software Configuration Management
What to Do When You Know You Are in Trouble

Software configuration management (CM) is a task often left until last. Consequently, motivation to do CM often comes from a fear of a project disaster rather than from faith in the control and visibility CM provides. This management by fear is akin to battlefield triage. Triage is a system to assign medical treatment priority to battlefield casualties based on urgency and chance for survival.

In the software battle, what will keep your project from bleeding to death? The answer is "stop-gap CM." Ted Gill's article is careful to point out that although stop-gap CM increases your project or organization's odds of survival over the short term, stop-gap CM will not produce the vibrant and healthy project or organization you need to compete for scarce Department of Defense funding year after year.

The Software Technology Support Center (STSC) does triage to see you through a CM crisis by assessing your situation, working with your leadership to plan a solution, and doing the necessary "hand holding" at the project level. And we do it with an eye toward the long-term health, stamina, and resilience your organization needs. On a cost-recovery basis, the STSC sees the job through to your desired state in which

- A CM organizational entity has been established.
- All CM policies and procedures are fully documented and implemented.
- The CM polices and procedures are tailorable to the size and scope of specific software development or maintenance projects.
- All software development or maintenance projects are conducted in compliance with CM policies and procedures.

The STSC – A practical approach to CM.

For more information, call
Paul Hewitt
Voice: 801-775-7775 ext. 3039
DSN 775-7775 ext. 3039
Reed Sorensen
Voice: 801-775-5555 ext. 3049
DSN 775-5555 ext. 3049
E-mail: scm@software.hill.af.mil

Everything beyond this level of documentation is window dressing. An easy way to approach documentation is to assemble the appropriate project team members (possibly with the software engineering process group [SEPG] CM process lead) and decide how to attack a particular part of the process. Once you have a method, create a memorandum that details the decisions and send it to all project members. Keep a copy of these memorandums in a CM process notebook. That is it—nothing fancy.

If a method does not work as it should, do not be afraid to change it. Just be sure to document the process change.

Following are examples of topics that should be documented:

CSCIs
- Lists of CSCIs and their baseline status.
- Levels of decomposition of CSCIs (what constitutes a configuration unit).
- Naming conventions and standards and version and revision identification.
- Configuration unit ownership lists and policies.

Changes
- Forms to be used and guidance on filling them out.
- Process flow descriptions for change submission and processing.
- Points of contact.

CCB
- Members.
- Duties and tasks, process flow descriptions.
- Meetings schedules.
- Record of actions (minutes).

Conclusion

This set of processes is not intended for the mature software development environment. It is aimed primarily at the organization that is taking those first difficult steps toward software process improvement. These processes serve as a consciousness-raising device as much as anything else. At the early stages, I have found that the two most essential ingredients to initiate process improvement were education aimed at people not steeped in the vocabulary and dogma of the Capability Maturity Model (CMM), and a plan of action that could be achieved by beginners. If you are in a situation where you have no configuration management, these processes provide the subject matter for education and a simple plan to execute. ◆

About the Author

Ted Gill is currently employed by the U.S. Navy at Puget Sound Naval Shipyard. He was the configuration management key process lead in the shipyard's SEPG. As a charter member of the group, he was involved with the initial efforts of assessing software practice, educating developers, and moving the shipyard's software development processes toward CMM Level 2. He served as project manager for the shipyard's training and qualification tracking system development project and has developed propulsion plant demonstrator software for the shipyard's nuclear training division. He now works in database administration and as a general process consultant for the shipyard's information resources management department.

Puget Sound Naval Shipyard
Code 1233
1400 Farragut Avenue
Bremerton, WA 98314-5000
Voice: 360-476-2072
Fax: 360-476-2275
E-mail: gillt@psns.navy.mil

NSWC PHD Dam Neck Receives CMM Level 3 Rating

On Sept. 19, 1997, The Naval Surface Warfare Center (NSWC) Port Hueneme Division (PHD), Dam Neck Detachment completed its external assessment to earn a Software Engineering Institute (SEI) Capability Maturity Model (CMM) Level 3 rating. NSWC PHD, Dam Neck is the first U.S. Navy tactical real-time program developer to attain Level 3.

The road to CMM Level 3 began in 1987 with an association between NSWC PHD, Dam Neck and the SEI, whereby the former served as a CMM development beta test site and an internal software process assessment reviewer. In 1992, NSWC PHD, Dam Neck continued down the software process improvement path by conducting a benchmark assessment using CMM as a guide. This assessment resulted in the development of a Software Process Improvement Plan, and training was provided to introduce all employees to CMM. The next step was completed in 1994 with the chartering of a full-time software process engineering group, which established a process asset library and developed the Standard Software Process Definition.

In 1995, NSWC PHD, Dam Neck reorganized along product lines, which allowed the new departments to focus on quality, process, and training. As a result, process improvement was institutionalized at the organizational level. NSWC PHD, Dam Neck continued with periodic internal assessments and achieved CMM Level 2 status in 1996. The final stop on the road to CMM Level 3 was achieved on Sept. 19, 1997 when an external assessment rated the directorate CMM Level 3. Throughout this entire evolution, NSWC PHD, Dam Neck has maintained its affiliation with the SEI.

Effective Software Configuration Management

Bob Ventimiglia
Lockheed Martin Aeronautical Systems

This article discusses effective software configuration management (CM) as the next step in the evolution of the configuration management practice. Effective CM stresses the technical aspects of CM, extends the concepts of classical CM, and meets the needs of both managers and developers.

In the beginning, there were rocks, dinosaurs, Fred Flintstone, and of course, classical CM, which was good for its time. But its time is now up. Its replacement is what I call "effective" CM. Effective CM is not your grandfather's CM. Effective CM is a nonadministrivia, nonclerical, integrated, cost-effective approach to implementing CM. In summary, effective CM is

- Cost-effective project insurance, i.e., product-integrity focused.
- A manager's and developer's best friend.
- A developer's tool, i.e., developer executed.
- A systems engineering function.
- Difficult to do right.
- No cost when done right (as part of daily development activities).

Effective CM is not

- Just version management.
- Just change management.
- Just a build-and-release function.
- An administrivia function.
- A trivial task.

The Institute of Electrical and Electronics Engineers (IEEE) definition of configuration management includes "a discipline applying technical and administrative direction and surveillance to. …" [1] The key words here are *technical, administrative,* and *surveillance*. Classical CM stresses administrative and surveillance and minimizes technical, which results in processes that developers see as obstacles to getting their job done. Effective CM stresses the technical while using tools to minimize the obstacles that can be caused by the administrative and surveillance activities. The IEEE definition further defines four functions performed by configuration management:

- Identification of configuration items.
- Change control of configuration items.
- Audit of configuration items.
- Recording and reporting of information needed to manage configuration items.

The International Organization for Standardization defines a configuration item as "a collection of hardware, software, and/or firmware, which satisfies an end-use function and is designated for configuration management." [2] The key part here is "which satisfies an end-use function and is designated for configuration management." Classical CM has typically defined configuration items as hard-copy documents and collections of completed source code. As this article will demonstrate, effective CM treats all files produced by the development process as configuration items.[1]

Classical CM focuses on the administrative direction and surveillance aspects of CM and downplays—in some cases ignores—the technical aspects of CM. Because it is labor intensive, classical CM applies an after-the-fact, reactive, "throw it over the wall when yer done" approach to CM for a relatively small number of configuration items. To many developers, this is an obstacle to the development process that adds little value.

Because of the lack of tools with which to manage configuration items, classical CM practitioners were and are highly detail-oriented people with a "green-eye-shades" mindset. I always envision Bob Cratchet in *A Christmas Carol* sitting with quill pen in hand, eyes protected by a green eye shade, slouched over Scrooge's account books, keeping detailed records of Scrooge's business transactions. Cratchet would make an excellent classical CM practitioner. Furthermore, the configuration items managed by classical CM are completed products such as documents and collections of source code with little or no support for in-process management of evolving designs and configurations or the environment used to generate them.

Businesses pressure us to do more with less and to speed development and change cycles. Changes in tool technology and new standards that represent world-class best practices (US 12207 and ISO 12207) require and support a revised approach to CM implementation—effective CM. Effective CM

- Is a system engineering function that is a pro-active, in-yer-face discipline that stresses the technical aspects of CM.
- Satisfies the needs of management as well as developers.
- Is unobtrusive to the point of being 100 percent developer executed.
- Is integral to the software development process.
- Supports the definition and implementation of the software development environment and process.
- Manages changes to all project components as they move through their development and approval cycles.
- Focuses on the "smallest work product of significance" to the development team.
- Follows the files and work products produced by the developer's tools.

Every file produced by the software development process can now be considered a configuration item. The number of objects managed by effective CM is orders of magnitude greater than those managed by classical CM. As a result, it carries the implicit requirement that it must be automated—automated by using CM tools that completely integrate process management (workflow), version, and change management, and that can act

Worth a Look

Configuration Management Readings

For further information about configuration management, the following sources may be useful.

Ben-Menachem, Mordechai, *Software Configuration Management,* McGraw-Hill, New York, 1994.

Berlack, Ronald, *Software Configuration Management,* John Wiley & Sons, New York, 1991.

Buckley, Fletcher J., *Implementing Configuration Management: Hardware, Software, and Firmware,* IEEE Computer Society Press, Los Alamitos, Calif., 1996.

Whitgift, David, *Methods and Tools for Software or Software Configuration,* John Wiley & Sons, New York, 1991.

Coordination for Team Productivity, *Software Configuration Management,* Addison-Wesley, Reading, Mass., 1986.

as the development environment and software repository.

Effective CM incorporates a paradigm shift that extends the concepts of classical CM to in-process management of software development work products, objects, entities, and artifacts, not just product-level documentation of product-level configuration items. It manages requirement identifications, not just requirement specification documents; elements of models as opposed to documentation that contains a completed model; interface messages instead of interface documents. Production of hard-copy documentation is avoided whenever possible by using the managed work products to document the system design. The level of control applied to each managed work product is based on the maturity level of that work product. As work products mature, change authorization migrates from author to software lead engineer to subsystem or system lead to program manager to customer. The level of control is kept to the lowest level practical throughout the products' lifecycle.

The effective CM process

- Supports and is built into the software development process.
- Is a core activity of the software development process.
- Helps manage the evolution of the software development work products.
- Focuses on management of the development process and on the work of and interactions between multiple developers.
- Applies a systems engineering view of CM to the software development process. (It is interesting to note that most industry and military standards identify CM as part of systems engineering.)
- Returns CM to its systems engineering roots and allows effective CM practitioners to act as systems engineers instead of clerks.
- Stresses the technical aspects of CM through an understanding of the development process and most important, by being part of the development process, by building CM into the development process, and by implementing tools that allow the developers to execute CM tasks with minimal intrusion into the development process.

Effective CM reinforces the belief that CM is good and that it is part of the solution, not part of the problem.

The systems integration approach to implementing effective CM begins with the development of a "living" software development plan (SDP). Effective CM practitioners participate in the creation of the SDP and the software development process to ensure that CM is built into the development process—a significant paradigm shift from the classical CM approach. Development methods are defined and a tool suite or system/software engineering environment (S/SEE) is selected that supports the development methods. CM participants ensure that the S/SEE includes tools to implement effective CM, e.g., completely integrated process, workflow, version, and change management CM tools that will be used by the developers. For effective CM to be successful, the CM practitioner must design for success by making the "right way" an "easy way" for developers to accomplish the tasks outlined in the SDP. One must never fail to recognize human nature. Developers will always take what they perceive to be the path of least resistance. Designed correctly, that way will be the effective CM way.

Summary

Because it requires less time on the clerical aspects than classical CM, effective CM allows a pro-active approach to maintain product integrity. It ensures the completeness and correctness of configured work products. It provides a stable environment in which to maintain the integrity of software development throughout the evolution of the project from concept to delivery to customers to product retirement. Its core focus is on problem avoidance and does not have to be cumbersome or complex. Effective CM eliminates the build difference problem where the engineering test build is different from the CM build. It eliminates the "wrong

file" in the build problem and other problems related to process breakdowns that manual (classical) CM systems frequently have. Effective CM is program management's eyes and ears into the project. It provides the who, what, where, when, and how and provides complete and accurate in-process software configuration status, state, and volatility of changes, and identifies areas that need management attention. Effective CM is the next step in the evolution of CM practice. ◆

About the Author

Bob Ventimiglia is an internationally recognized expert in state-of-the-science CM. He is currently the environment, tools, and CM lead for the software engineering process department at Lockheed Martin Aeronautical Systems in Marietta, Ga. He leads the effort to enhance and evolve the Hercules C-130 program into an effective CM process. He has been the F-22 environmental control systems and brake control systems software manager, lead F-22 CM engineer, and chairman of the F-22 CM process configuration management system working group. Prior to joining Lockheed Martin, he held key software management and engineering positions with GE Aerospace Information Technology, GE Corporate Engineering and Manufacturing, Sanders Associates, Inc., and Hamilton Standard Division of United Technologies. He has a master's degree in engineering mechanics and a bachelor's degree in aeronautics and astronautics from New York University School of Engineering.

Lockheed Martin Aeronautical Systems
86 South Cobb Drive
Marietta, GA 30063-0685
Voice: 770-494-9791
Fax: 770-494-1345
E-mail: bventimi@hercii.mar.lmco.com (office)
bobev@technologist.com (home)

References

1. IEEE STD 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology."
2. ISO 10007:1995 (E), "Quality Management – Guidelines for Configuration Management."

Note

1. A more extensive list of definitions of CM—including those by Susan Dart that are coming close to a modern redefinition of CM—can be found at http://www.enteract.com/~bradapp/acme/.

Configuration Management Web Sites

Yahoo configuration management (CM) links
http://www.yahoo.com/Computers_and_Internet/Software/Programming_Tools/Software_Engineering/Configuration_Management

CM Yellow Pages (provided by André van der Hoek)
http://www.cs.colorado.edu/users/andre/configuration_management.html

Brad Appleton's home page and CM links
http://www.enteract.com/~bradapp

Software Technology Support Center (STSC) home page
http://www.stsc.hill.af.mil

Software Engineering Institute CM home page
http://www.sei.cmu.edu/technology/case/scm/scmHomePage.html

CM frequently asked questions from the Usenet CM group
http://www.iac.honeywell.com/Pub/Tech/CM/CMFAQ.html

Other Web Sites of Interest

CROSSTALK
http://www.stsc.hill.af.mil/CrossTalk/crostalk.html

Association for Configuration and Data Management
http://www.acdm.org

Electronic Industries Association
http://www.eia.org/gd

Managing Standards home page
http://www.airtime.co.uk/users/wysywig/wysywig.htm

Data Interchange Standards Association
http://www.disa.org

International Organization for Standardization
http://www.iso.ch/welcome.html

Software Productivity Research
http://www.spr.com

Official Department of Defense Stock Point
http://www.dodssp.daps.mil

Software Deployment

Extending Configuration Management Support into the Field

André van der Hoek, Richard S. Hall, Antonio Carzaniga,
Dennis Heimbigner, and Alexander L. Wolf
University of Colorado

Traditionally, configuration management has only addressed the needs of the software development process. Once a software system leaves the development environment and enters the field, however, there still is a significant role for configuration management. Activities such as release, installation, activation, update, adaptation, deactivation, and de-release constitute the "deployment lifecycle"; these activities need careful coordination and planning in their own right. This article discusses the dimensions of software deployment, argues why current solutions are not sufficient, and presents two research systems that specifically address software deployment.

The management of software systems after they have been deployed is an emerging problem that manifests itself in numerous places. Software on the Mars Pathfinder rover Sojourner was regularly changed by mission control to give it new behavior in the exploration of Mars. During the Persian Gulf war, software on Patriot missiles stationed in Israel and Saudi Arabia was updated with U.S.-implemented patches to increase their ability to intercept hostile Iraqi missiles. Also, software for on-board military aircraft computers is continually adapted to compensate for the various missiles carried on each mission.

All these activities are part of the software deployment process, which is defined as

> The *delivery*, *assembly*, and *maintenance* of a particular *version* of a *software system* at a *site*.

The elements of the above definition can be illustrated in more detail using the Mars Pathfinder as an example. The *site* in this case is the rover, whereas the *software system* is the software that controls the movement of the rover over the planet as well as the operation of its measuring instruments. The *delivery* is the transmission from Earth of new code or new data to the Pathfinder, whereas the *assembly* process ensures consistency in the inclusion of new code or data in the system already present at the rover. This results in multiple *versions* of the software system to be present at the rover, which need to be *maintained* by ground control.

At first sight, the software deployment processes for the Mars Pathfinder, Patriot missiles, and military aircraft seem vastly different. However, a significant amount of commonality exists among these and many other deployment processes. This article examines this commonality. We first define the software deployment lifecycle, which consists of the series of activities normally carried out during the deployment of a software system. We then examine some existing solutions and demonstrate why these solutions are not sufficient to solve all deployment problems. We conclude with a brief look at two research prototypes we have been constructing that provide a radically different approach to software deployment.

This work was supported in part by the Air Force Materiel Command, Rome Laboratory, and Defense Advanced Research Projects Agency (DARPA) under contract number F30602-94-C-0253, and in part by DARPA and the Office of Naval Research (ONR) under ONR contract number N66001-95-D-8656, delivery order 0001. The content of the information does not necessarily reflect the position or the policy of the U.S. government and no official endorsement should be inferred.

Software Deployment Lifecycle

A software system's general deployment process is composed of a variety of subprocesses or activities. Figure 1 lists these activities and organizes them into an overall deployment lifecycle. Following is a more detailed discussion of each activity.

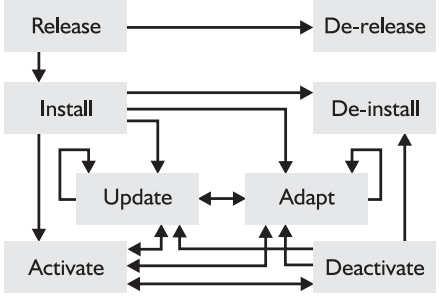
Release

The release process is the interface between the development process and the deployment process. It encompasses all activities needed to prepare and advertise a system so that it can be assembled correctly at another site. The result of this activity is a package that contains the system components, the systems dependencies and constraints, and information needed for the other deployment steps. In addition, this package is advertised.

Install

The installation activity covers the initial insertion of a system onto a site. It is usually the most complex of the deployment activities because all the necessary resources must be found and assembled. In the installation process, the package created in the release process is used, then the encoded knowledge is interpreted and the target site is examined to determine how to properly configure the software system.

Figure 1. *Software deployment lifecycle.*



		Package	Advertise	Transfer	Configure	Activate	Transfer	Reconfigure	Adapt	Deactivate	De-install	De-release
		Release		Install		Activate	Update		Adapt	Deactivate	De-install	De-release
Content Delivery	Castanet		o	•			•					
	PointCast		•	•								
	Rsync			•			•					
	Rdist			•								
Install and Update	NetInstall	o		o	o						•	
	NetDeploy	o		o	o		o	o			•	
	InstallShield	o			•						•	
	RPM	•		o	•		o	•	o		•	
	HP-UX SD	•			•			•	o		•	
Standards	MIF											
	AMS											
	Autoconf				o			o				
Network Mgt.	TME-10			•	•	•	•	•	o	•	•	
	Platinum			•	•	•	•	•	o	•	•	
	Nebula			•	•	•	•	•		•	•	
	Novadigm				•	•		•		•	•	

Table 1. *Evaluation of software deployment lifecycle support coverage. The release, install, and update activities have been split into two subactivities to better highlight system features. "*o*" indicates some support, "*•*" is better-than-average support.*

Activate

Activation refers to starting up those components of a system that must execute for the system to be usable. For large systems in particular, activation can be especially complex because it involves the start-up of servers, clients, database systems, etc.

Deactivate

Deactivation is the inverse of activation, and refers to shutting down any executing components of a system. Deactivation is often required to perform other deployment activities, e.g., a software system may need to be deactivated before an update can be performed.

Update

The update process involves modifying a software system that has been previously installed on a site. An update is a new version of a software system that fixes a bug or adds new functionality. Updates are normally less complex than installations because many of the needed resources have already been obtained during the installation process. Although a system is usually deactivated before an update, this is not always the case. For some systems, there is a stringent requirement that they continue to operate while being updated. In such cases, the update process can be quite complex.

Adapt

The adapt process involves modifying a software system that has been previously installed at a site. Adapt differs from update in that updates are instigated by remote events, whereas adaptations are instigated by local events. For example, if the configuration of a site changes in a way that affects the deployed system, it may be necessary for the deployed software system to take corrective action.

De-install

At some point, a system as a whole is no longer required at a site and can be de-installed. De-installation is not necessarily a trivial process. Special attention has to be paid to shared resources such as data files and libraries to prevent dangling references.

De-release

Ultimately, a system may be marked obsolete, and support by its producer is withdrawn. De-release is distinct from de-installation in the sense that the software system becomes unavailable for further installation at sites, but it is not removed from sites that are using the software.

Classification

The recent emergence of Internet-based deployment systems has created a renewed interest in software deployment. However, solutions for software deployment problems have been around for decades, and many deployment systems exist that support one or more activities of the deployment lifecycle. These existing systems can be categorized into some combination of the four classes discussed below.

Content Delivery

In this class of systems and technologies, the information being deployed is merely transferred from one or more sources to a number of receiving sites. No customization is carried out once this information has been placed at a site. In essence, content delivery systems provide a replication mechanism between source and target sites.

System Install and Update

These systems deal with the localization of a software system to the environment

provided by a site. Both an initial localization and incremental updates are supported. Most deployment systems fall into this class.

Standardization

Because of the large amount of information that needs to be managed during deployment of a software system, it is not surprising that some efforts for standardization have taken place. Standards generally focus on creating a standard template to describe software systems and deployment sites. These templates are used by deployment systems to manage the deployment process.

Network Management

In the past, these systems have dealt with managing hardware systems in a heterogeneous network. Over time, they have grown to include some deployment activities, such as installation, update, and de-installation. The systems in this class often operate in a centralized notion; they assume "dumb" target sites.

Current Solutions

We have examined a representative set of systems from each of the above classes with respect to coverage of the deployment lifecycle. The results of this evaluation can be found in Table 1. However, such a simple examination is not sufficient to fully characterize deployment systems. We also have to look at several other characteristics of deployment systems.

We strongly emphasize the importance of the changeability and parameterization of the deployment process embodied in deployment systems. This requires deployment systems to operate at a certain level of abstraction. The primary abstractions are the target site, the system to be deployed, and the process. As more powerful modeling capabilities for these abstractions are included in a deployment system, two benefits arise. First, simple systems can be deployed using generic processes. Second, more complex software systems can be successfully deployed. For example, content delivery systems often do not model the software system to be deployed, whereas system installers have knowledge about the composition of a software system. Therefore, updates done by content delivery systems are often based on an algorithmic difference, but updates done by systems installers are mostly based on a firsthand understanding of the components that need to be changed.

Another important aspect to examine is support for the coordination of distributed, cooperating software systems. These systems contribute to the growing complexity of software deployment because their architectures have inherently complex, unreliable relationships and dependencies. Such architectures require special support to deploy successfully because coordination among servers, peers, and clients may be necessary. These coordination issues complicate the activities of the software deployment lifecycle.

Table 2 presents the results of the second part of our examination. The table reflects the ability of each examined deployment system to model, change, and parameterize the target site, the system to be deployed, and its embodied process. It also presents each system's ability to support the deployment of distributed, coordinated systems.

		Site Abstraction	System Abstraction	Process Abstraction	Coordination
Content Delivery	Castanet				
	PointCast				
	Rsync				
	Rdist				
Install and Update	NetInstall		o		
	NetDeploy		o		
	InstallShield		o		
	RPM	o	•	o	
	HP-UX SD	o	•	o	
Standards	MIF	•	o		
	AMS	•	•		
	Autoconf		o		
Network Mgt.	TME-10	•	•		•
	Platinum	•			•
	Nebula	•	•		•
	Novadigm	•	•		•

Table 2. *Evaluation of abstraction and coordination capabilities.* "o" *indicates some support,* "•" *is better-than-average support.*

University of Colorado Approach

As Tables 1 and 2 show, none of the existing deployment systems support the entire software deployment lifecycle. More important, support for appropriate abstraction and coordination is lacking in most systems. What is needed is an all-encompassing, highly parameterized, unifying approach to software deployment. Because it is unlikely that ad hoc combinations of existing systems would yield the desired result, the Software Engineering Research Laboratory (SERL) at the University of Colorado at Boulder, funded by the Defense Advanced Research Projects Agency (DARPA)-sponsored Evolutionary Design of Complex Software (EDCS) program is researching software deployment systems. Two prototype systems have been created, each of which is briefly described below.

SRM – A Software Release Manager

Software Release Manager (SRM) focuses on the release activity of the deployment process. It supports the release of systems of systems from multiple, geographically distributed organizations. In particular, SRM tracks dependency information to automate and optimize the retrieval of components. Both developers and users of software systems are supported by SRM. Developers are supported by a simple release process that hides distribution. Users are supported by a simple retrieval process that allows the retrieval, via the Web, of a system of systems in a single step as a single package.

New Software Engineering Mailing List

The Software Engineering Research Laboratory at the University of Colorado at Boulder invites you to subscribe to a new, noncommercial mailing list for the software engineering community: SEWORLD@cs.colorado.edu.

SEWORLD will serve as a central place for relevant announcements of software engineering conferences, workshops, symposiums, special journal issues, calls for papers, research and educational systems, etc.

The list is moderated to avoid junk E-mail, duplication, and other misuses. In addition, all E-mail addresses are registered privately to the list, are not published, and will not be given to anyone requesting them.

Simple instructions on how to subscribe or contribute to SEWORLD are on the SEWORLD Web site at

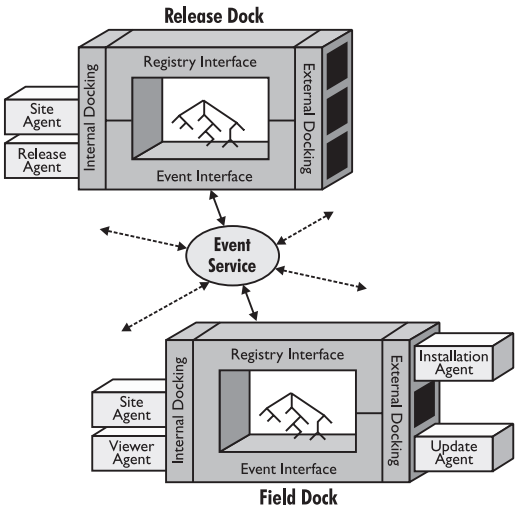
http://www.cs.colorado.edu/serl/seworld

SRM is freely available and supported on most UNIX platforms. It has been used extensively by our research laboratory to release our software and is currently being deployed to the approximately 50 participants in the EDCS program. This will provide the EDCS program with a single location from which all technology created by EDCS can be browsed and retrieved, despite that the technology is created by organizations spread over the United States.

Software Dock – A Software Deployment Architecture

Extending the ideas of SRM, the Software Dock constructs an architecture that supports *all* of the activities in the software deployment process. The Software Dock relies not only on a standard available *release dock* at a producer site (similar to SRM), but also on a standard available *field dock* at a target site (see Figure 2). Between these docks, *agents* accompany software systems as they are being deployed. The agents represent the various deployment activities and operate on the semantic data (available in the registries of the release and field docks) to properly deploy a software system. Standard agents are available that model most common deployment activities. These agents can be parameterized to carry out more specific deployment activities as required.

Figure 2. *Software Dock architecture.*



Central to the Software Dock approach is that all dimensions are customizable. In particular, the release dock is the abstraction for a software system to be deployed, the field dock is the abstraction for the target site, and the agents are the abstraction for the deployment process activities.

A prototype of the Software Dock is currently being implemented. Previous incarnations have been created, evaluated, and constructed based on a collaboration with Lockheed Martin, wherein an existing 6,000-line Perl-based installation script was reduced to a small installation agent derived from a generic installation template.

Conclusion

Although software deployment is an ever-present activity as software systems are being developed, structured support for the deployment process has been remarkably lacking until now. Various deployment systems have been created, but no system comes close to providing a single solution that can "do it all." Based on a well-defined deployment lifecycle, we have highlighted some of the complex issues, partial solutions, and research to define and build the software deployment systems of tomorrow. ◆

Further Reading

Space limitations do not allow us to treat all issues in as much depth as they deserve. Please visit the following Web sites for further information on deployment issues, existing deployment solutions, and the prototypes discussed in this article.

- SRM http://www.cs.colorado.edu/serl/cm/SRM.html.
- Software Dock http://www.cs.colorado.edu/serl/cm/dock.html.
- The Configuration Management Yellow Pages http://www.cs.colorado.edu/users/andre/configuration_management.html.
- The Software Deployment Information Clearinghouse http://www.cs.colorado.edu/users/rickhall/deployment/.
- SERL http://www.cs.colorado.edu/serl.

About the Authors

André van der Hoek is a computer science doctoral candidate at the University of Colorado at Boulder. He has a bachelor's degree and a master's degree in business-oriented computer science from

the Erasmus University Rotterdam, the Netherlands. His research interests include configuration management, software architecture, and distributed systems. He is a member of the program committee of the Eighth International Symposium on System Configuration Management.

Richard Hall is a computer science doctoral candidate at the University of Colorado at Boulder. He has a bachelor's degree in computer engineering from the University of Michigan and a master's degree in computer engineering from the University of Colorado at Boulder. His research interests include software deployment and distributed systems. He currently works on a distributed, agent-based framework to support software deployment.

Antonio Carzaniga has a laurea degree in electronic engineering from Politecnico di Milano and a master's degree in information technology from CEFRIEL, Milano, Italy. He was a junior researcher with

CEFRIEL before entering the doctorate program in software engineering at Politecnico di Milano. Currently, he is a visiting research assistant in the computer science department of the University of Colorado at Boulder. His interests include software process as well as generic distributed systems technology. His current research concerns scalable event observation and notification mechanisms.

Dennis Heimbigner has a bachelor's degree in mathematics from the California Institute of Technology and a master's degree and a doctorate in computer science from the University of Southern California. He is a former member of the technical staff for TRW Defense and Space System Group in Los Angeles, Calif. He is currently a research associate and assistant professor in the computer science department of the University of Colorado at Boulder.

Alexander Wolf is a faculty member in the computer science department of the University of Colorado at Boulder. Previously, he was employed at AT&T Bell Laboratories. His research interest is the discovery of principles and development of technologies to support the engineering of large, complex software systems. He has published papers on software engineering environments and tools, software process, software architecture, and configuration management. He is vice chairman of the Association for Computing Machinery Special Interest Group on Software Engineering.

Software Engineering Research Laboratory
Department of Computer Science
University of Colorado
Boulder, CO 80309
Voice: 303-492-5263
Fax: 303-492-2844
E-mail: {andre, rickhall, carzanig, dennis, alw}@cs.colorado.edu

Coming Events

Second Workshop on Software Architectures in Product Line Acquisitions

Dates: June 8-10, 1998
Location: Salem, Mass., Hawthorne Hotel
Subject: Adoption of an architecture-driven approach to acquiring a line of software-intensive products.
Call for Position Papers: Submissions due: March 6, l998. Notification of acceptance: April 1, 1998. For submission guidelines, visit the Production Line Issues Action Team Web site.
Sponsor: Product Line Issues Action Team
Contact: Edward Addy, NASA/WVU Software Research Laboratory
Voice: 304-367-8353
Fax: 304-367-8211
E-mail: eaddy@wvu.edu
Internet: http://columbia.ivv.nasa.gov:6600/pliat

Relationship of DoD Architecture-Driven Standards to Product Line Acquisition Business Model

Dates: March 17-18, 1998
Location: Burlington, Mass.
Sponsor: System Resources Corporation
Subject: Meeting participants will discuss the applicability of Department of Defense (DoD) architecture initiatives to software architecture-based acquisitions of a product line and produce a short point paper discussing how product line acquisition organizations can effectively apply DoD standards in the acquisition of a family of software-intensive systems.
Contact: Harry Joiner
E-mail: hjoiner@world.std.com

International Information Technology Quality Conference

Dates: April 13-17, 1998
Location: Orlando, Fla.
Theme: "Providing Proven Solutions for the New Millenium"
Keynote Speakers: Phillip Crosby, Tom DeMarco, William Perry, Howard Rubin
Sponsor: Quality Assurance Institute
Contact: 407-363-1111
Fax: 407-363-1112
Internet: http://www.qaiusa.com

Three Dimensions of Process Improvement
Part I: Process Maturity

Watts S. Humphrey
Software Engineering Institute

This is the first part of a three-part article on methods of software process improvement that were developed at the Software Engineering Institute (SEI): the Capability Maturity Model (CMM)® for software, the Personal Software Process (PSP)℠, and the Team Software Process (TSP)℠. The CMM provides an overall framework to describe the activities software organizations need to do to consistently produce effective results; the PSP helps engineers use process principles in their personal work; the TSP shows integrated product teams how to use these processes to consistently produce quality products on aggressive schedules and for their planned costs. Each method provides important benefits; organizations will get the best results by using all three. Part I describes the CMM. Parts II and III (to appear in subsequent issues) will cover the PSP and TSP.

The first issue to be addressed in any improvement program is "Why should we improve?" Regardless of how logical the improvement is and regardless of growing evidence of its benefits, organizations do not launch successful process improvement programs until they have a compelling business reason.

One of the most effective improvement motivators is customer pressure. In the case of the CMM, the U.S. Air Force asked the Software Engineering Institute (SEI) to devise an improved method to select software vendors. When we responded with the CMM, the defense industry then had a compelling business reason to improve their software processes. As the CMM became more widely used, the business logic for improvement also evolved. Now, many software organizations desire to establish and maintain a strong competitive position as quality suppliers of software-intensive systems.

Once the "why" question is answered, the next immediate question is "What must we do to achieve a superior software capability?" This is the principal question addressed by the CMM. The maturity-level framework and related evaluation system help organizations understand their capabilities. They can then compare their current practices with the CMM model and see what activities they need to add or improve.

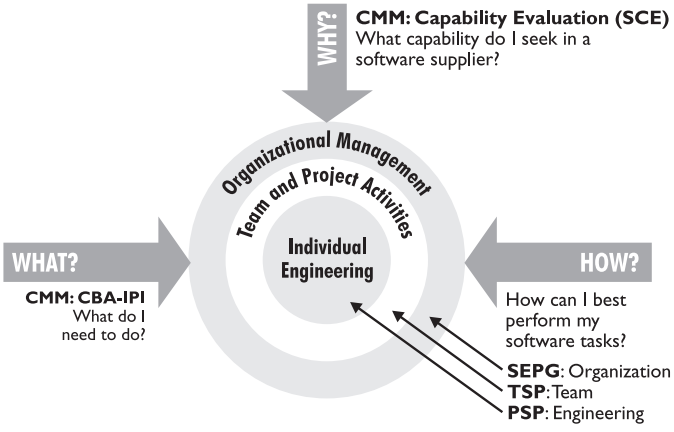
Once an organization knows what to do to improve, the next question is "How do we make these improvements?" This question involves several organizational levels. At the management level, a software engineering process group (SEPG) helps establish the definition, control, and improvement tasks needed to launch an improvement program. At the next level, the TSP guides project teams and their management in applying process principles to meet project objectives. Finally, the PSP addresses the way engineers develop products. For the organization to meet its objectives, the engineers and managers at all these levels must perform their tasks with skill and competence. In the last analysis, to do superior software work, organizations must have high-performance software engineers working on well-managed and high-performing project teams.

Figure 1 shows the relationships among these management, team, and engineering layers. Organizational capability is built from the inside out while project support and engineering motivation are provided from the outside in. Effective work in all three layers is required for a fully productive software organization. The balance of this article describes why we developed the CMM, PSP, and TSP, and how they work.

Why Software Projects Fail

There are many ways to botch up a software project: have ineffective management, execute poor engineering, or create a confusing design. To be consistently successful, competent work must be done in all important technical and management areas. If only one area is not addressed—requirements management, for example—that area could easily become the Achilles heel that results in the project's failure. (Not to say that requirements management is more important than all the things done correctly, just that through neglect it became the cause of the failure). The next time around, requirements may

Figure 1. *Process improvement dimensions.*



The SEI's work is supported by the Department of Defense.

be managed properly but the design botched, and another project fails.

Although the cause of failure may not matter to the ultimate user, the reasons are important from a process improvement perspective. If the causes are managerial or administrative, improved organizational and project management will usually solve the problems. If the causes are purely technical, however, the solutions are generally more complex.

A careful examination of failed projects shows that they often fail for nontechnical reasons. The most common problems concern poor scheduling and planning or uncontrolled requirements. Poorly run projects also often lose control of changes or fail to use even rudimentary quality practices.

The SEI Process Improvement Approach

The SEI started its process improvement work by first attacking project management problems, because poor management practices are the most frequent cause of software project failure. When the project is poorly managed, it is hard to improve anything else.

Again, it is important to remember that all aspects of the project are important, and any omitted area could be fatal. Thus, while an organization's improvement work focuses on management issues, the development groups must continue to do their best to address all the important facets of their work.

Focusing the Improvement Effort

Process change is behavioral change and behavioral change takes a great deal of time. Behavior involves long-held habits and practices that people will not readily abandon. If change is imposed by force, people will resist what they perceive to be a threat to their stability. The most effective way to avoid resistance is to convince people that the change is in their best interest; however, it may take a long time for some people to be persuaded. Further, to have any hope of a lasting change, only a few changes should be addressed at a time. With several dozen critical issues, for example, making them all high priority would be confusing, and when confused, people are much less likely to change the way they work.

The only way to initiate process improvement is to look at the problems in *your* organization. No two organizations are identical, and you need to determine where your group can most productively focus its improvement energies. Figure out which of these problems are fundamental and which will make the most difference. Then, select those few changes that will most directly address the current needs of your business and people.

The CMM process maturity road map and the CMM-based appraisal for internal process improvement (CBA-IPI) helps organizations evaluate their particular weaknesses [1]. By doing a CBA-IPI assessment, organizations can both identify the most important areas for improvement and establish an organizational consensus on the need for improvement. This helps organizations set improvement priorities and reduces resistance to change [2, 3].

The Motivation for the CMM

When I arrived at the SEI in 1986, I was asked to work on a high-priority Air Force project. The military's experience with software acquisition had been, in summary, less than satisfactory. This "software crisis" was the principal reason the U.S. Department of Defense (DoD) established the SEI in the first place. Thus, one of our first projects was to devise a method the DoD acquisition community could use to identify competent software contractors.

We knew from many years of software experience that management attention is the key to process improvement. Without sufficient management priority, not much improvement work gets accomplished, at least not for a long time. Management, however, is generally sensitive to their customers' demands. Thus, if the DoD acquisition community required improved software processes from their suppliers, we were sure management would give the subject a high priority. The original motivation for the CMM was thus to address DoD's problems with software acquisition.

The Birth of the CMM Concept

The original concept behind the CMM started to gel in my mind many years ago when I was put in charge of a large software development organization. This group had several thousand engineers working on many large and small projects in 15 laboratories in the United States and Europe. Almost all the projects were in trouble and nothing was on schedule.

My first step was to visit several of the largest of these development laboratories. Nobody had plans or schedules. They all agreed that to deliver products on time, they needed to have schedules and plans. They also agreed that if they did not make plans, they could never make sensible commitments. And without sensible commitments, they could not start delivering on schedule. They said the reason they could not make plans and schedules was that if they put anyone on the planning work, they would have to take them from development, which would delay the projects.

So none of the laboratories made plans and schedules. Their commitments were pure guesses, and they were almost never met. This had been going on since the laboratories were formed, and it appeared likely to continue unless something happened to cause change. My question was if everybody agreed that planning was essential, why didn't they do it?

It Was My Fault

The answer was that I was not doing *my* job. These laboratories had so much to do and they were under so much pressure that they could only do what they absolutely had to do to ship the products. In simplest terms, that meant coding and testing; everything else was considered unnecessary and could be skipped. As long as the laboratories could announce and ship products without planning, they would continue to do so.

Merely telling the laboratories to make plans would not work. So I put out a directive that henceforth, no prod-

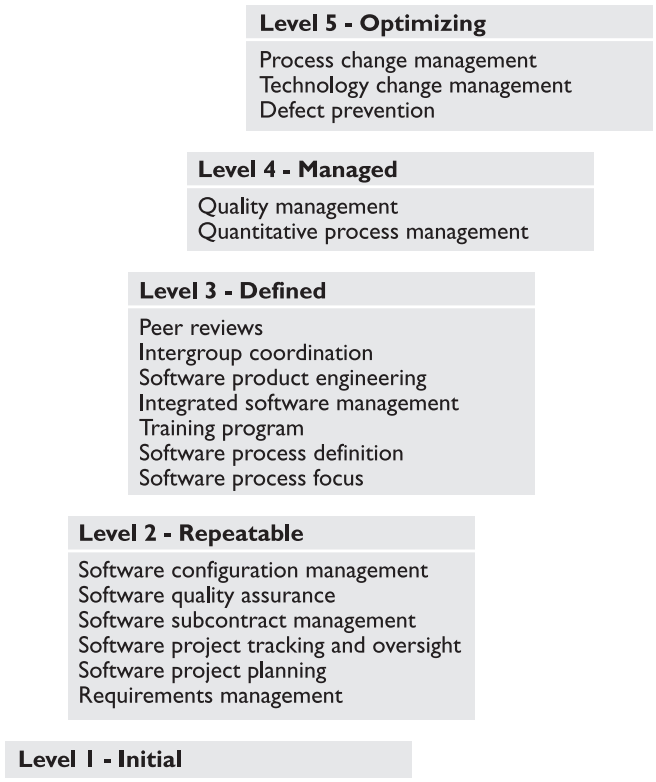


Figure 2. *CMM maturity levels.*

uct would be funded, committed, or shipped unless I first had a documented plan on my desk. This plan had to be signed by all the managers whose groups were involved in the work. I gave the laboratories 60 days to complete the first plans. Once we had plans for the work, we announced the new schedules, and we did not miss a single shipment date for the next two and one-half years.

The First Important Lesson

Effective software process improvement will not start until management insists that product development work be planned and properly managed. Senior management must then continue to insist that all the projects be planned, even in a crisis. They not only must insist on aggressive plans, they must respect the plans and ensure that the engineers own and defend them. Since the schedules engineers initially make are almost always too tight, management must review and critique these plans to look for holes and oversights. While management should push as hard as they can for aggressive schedules, once the engineers have defended their plans, they should respect these plans and not override them with schedule edicts.

We had to start at the top with a management commitment to planning. The engineers could not take that step themselves. As long as senior management let them slide by without plans, they would continue to do so. Of course it was not quite that simple. The engineers needed a lot of help with their first plans, and we had to develop a project management training program. Over the next three years, we put 1,000 managers through a one-week project management course.

Building the CMM Model

When I arrived at the SEI, we started to think about the improvement process in a more orderly way. We realized that process improvement must be taken in steps not only because people can change just a few things at a time but also because some steps are prerequisites for others. Since poor project management generally blocks good engineering, the first priority must be effective project management.

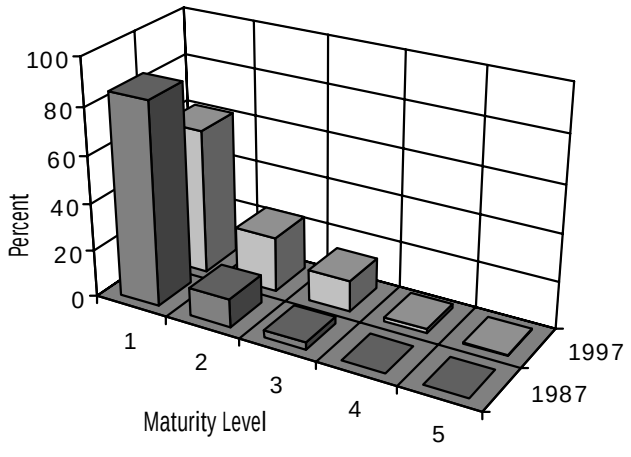
In addition to planning and scheduling, other tools that aid process improvement are quality assurance, configuration management, subcontract management, and requirements management. Quality assurance provides the management eyes and ears to ensure that the plans are properly made and that commitment ground rules are followed. Configuration management and subcontract management track and manage the products as they are developed, and effective requirements management maintains control of job scope. As shown in Figure 2, these are the CMM Level 2 practices [2, 3]. In essence, they establish a planned and managed environment that enables good engineering.

The Higher Maturity Levels

Beyond basic management, the next step is to address organizational learning and growth. As long as organizations learn only from their own mistakes, they can expect to make a lot more mistakes. Most software organizations have many good practices that are only used by a few projects. An organization can make rapid improvement when these practices are more widely used. The objective is to capture those effective practices so others can use them, which leads to what we call process definition. Although there are many more parts to CMM Level 3, its essence is to facilitate organizational learning by getting good practices defined and into use.

The next steps focus on process measurements, quality management, and quality control. Organizations also need to introduce advanced methods and technologies. Finally, the

Figure 3. *State of software practice 1987-1997.*



Yearly cost of improvement	$245,000
Years engaged in improvement	3.5
Yearly cost per software engineer	$1,375
Productivity gain per year	35%
Yearly reduction in time to market	19%
Yearly reduction in post-release defects	39%
Business return per dollar invested	$5

Table 1. *Median results from an SEI study of 13 software organizations.*

software business is new, and we can expect it to continue to grow and evolve. Since any static process will soon be outdated, software organizations need to continue to improve. This calls for a continuing focus on technology change and process improvement. In total, these are the practices of CMM Levels 4 and 5.

Introducing and Using the CMM

Once management is convinced of the business need for process improvement, the most effective way to start is to achieve a broad consensus on the need for and potential benefits of process improvement, which is the role of the CBA-IPI assessment. The SEI trains people on how to conduct such assessments, and many organizations now offer commercial assessment services. These offerings are described more fully on SEI's Web pages http://www.sei.cmu.edu.

Conclusion

A large number of organizations have used the CMM framework, and a great deal has been published on its benefits [4,5,6,7,8,9,10,11]. Table 1 shows a brief summary of improvement data from one SEI study [12]. These 13 organizations had worked on software process improvement for an average of three and one-half years, and they all gained substantial cost, schedule, and quality benefits. Figure 3 shows the improvement in CMM levels between 1987—when SEI gathered initial data on a few dozen groups—and the most recent 1997 canvass of about 600 organizations. Although much more remains to be done, the CMM helps organizations improve. ◆

About the Author

Watts S. Humphrey is a fellow at the SEI of Carnegie Mellon University, which he joined in 1986. At the SEI, he established the Process Program, led initial development of the CMM, introduced the concepts of Software Process Assessment and Software Capability Evaluation, and most recently, the PSP and TSP. Prior to joining the SEI, he spent 27 years with IBM in various technical executive positions, including management of all IBM commercial software development and director of programming quality and process.

He has master's degrees in physics from the Illinois Institute of Technology and in business administration from the University of Chicago. He is the 1993 recipient of the American Institute of Aeronautics and Astronautics Software Engineering Award. His most recent books include *Managing the Software Process* (1989), *A Discipline for Software Engineering* (1995), *Managing Technical People* (1996), and *Introduction to the Personal Software Process* (1997).

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Voice: 412-268-6379
E-mail: watts@sei.cmu.edu

References

1. Dunaway, Donna and Steve Masters, "CMM-Based Appraisal for Internal Process Improvement (CBA-IPI): Method Description," Technical Report CMU/SEI 96TR-007, April 1996.
2. Humphrey, W. S., *Managing the Software Process*, Addison-Wesley, Reading, Mass., 1989.
3. Paulk, Mark C., et al., *The Capability Maturity Model: Guidelines for Improving the Software Process,* Addison-Wesley, Reading, Mass, 1995.
4. Diaz, Michael and Joseph Sligo, "How Software Process Improvement Helped Motorola," *IEEE Software*, Vol. 14, No. 5, Sept. 19, 1997, pp. 75-81.
5. Dion, Raymond, "Process Improvement and the Corporate Balance Sheet," *IEEE Software*, July 1993, pp. 28-35.
6. Goldenson, Dennis R. and James D. Herbsleb, "After the Appraisal: A Systematic Survey of Process Improvement, Its Benefits, and Factors that Influence Success," Technical Report CMU/SEI-95-TR-009, August 1995.
7. Hayes, Will and Dave Zubrow, "Moving On Up: Data and Experience Doing CMM-Based Process Improvement," Technical Report CMU/SEI-95-TR-008, August 1995.
8. Humphrey, W.S., T.R. Snyder, and R.R. Willis, "Software Process Improvement at Hughes Aircraft," *IEEE Software*, July 1991, pp. 11-23.
9. Lawlis, P.K., R.M. Flowe, and J.B. Thordahl, "A Correlational Study of the CMM and Software Development Performance." *CROSSTALK*, STSC, Hill Air Force Base, Utah, September 1995, pp. 21-25.
10. Wohlwend, H. and S. Rosenbaum, "Schlumberger's Software Improvement Program," *IEEE Transactions on Software Engineering*, Nov. 11, 1994, pp. 833-839.
11. Yamamura, George and Gary B. Wigle, "SEI CMM Level 5: For the Right Reasons," *CROSSTALK*, STSC, Hill Air Force Base, Utah, August 1997, pp. 3-6.
12. Herbsleb, J.D., D. Zubrow, D. Goldenson, W. Hayes, and M. Paulk, "Software Quality and the Capability Maturity Model," *Communications of the ACM*, June 1997, Vol. 40, No. 6, June 1997.

Process Action Teams
From "Black Holes" to "Shining Stars"

Douglas D. Orville
pragma Systems Corporation

Often because of unfortunate experiences, many people desperately avoid being selected for a process action team (PAT). No one wants to waste scarce time participating on an unproductive "team." Participants want to know what they are expected to produce, when they must produce it, who wants it and why, and how they are supposed to do whatever a PAT does. There is a better way to form PATs. This article addresses where PATs fit in the big picture of process improvement, who should be on a PAT, and how a PAT can create a defined and documented process. Results, such as the number of PAT sessions required and the estimated time for PAT activities, are provided.

Senior leaders of many organizations form PATs or other similar working groups to improve processes. These same leaders often believe that the business processes at the heart of the organization must be meticulously well defined and documented. It is often believed that all the PAT must do is quickly analyze the process and make some improvements.

Unfortunately, the opposite is usually true. Where any process documentation exists, it is often sketchy. Frequently, important information is not written down at all but instead resides in the heads of a few people. Often, the disappointing outcome is either a black hole—a PAT that goes on forever collecting input and thrashing about without producing any useful output—or a PAT that is terminated after management tires of waiting for results.

Process improvement by definition is impossible without defined and documented processes to improve. Unfortunately, most PATs do not know how to define and document a process. If there were a process that PATs could follow and assistance to follow the process, PATs could become shining stars that save significant amounts of time and effectively perform the task of defining and documenting critical business processes. The processes could then enter into the continuous process improvement cycle.

Where Do PATs Fit?

A variety of groups are often formed within an organization to improve processes, including a management steering committee (MSC), a software engineering process group (SEPG), and PATs. Pilot projects are used to test the new processes to ensure they are "fit for use" before the MSC publishes policies that mandate the use of the new processes by other projects in the organization. Figure 1 depicts a simplified view of some of the most important relationships. This article focuses on the relationship between the PAT and the SEPG.

Unless PAT members have experience defining and documenting processes, which is a specialized skill, the PAT will need the help of a group with such expertise. As depicted in Figure 1, the SEPG should have process documentation expertise gained from working with PATS. If not, the SEPG needs training and coaching in the use of such a process before they can guide the PATs. When SEPG training and coaching is not feasible, an outside organization with the required expertise should be brought in.

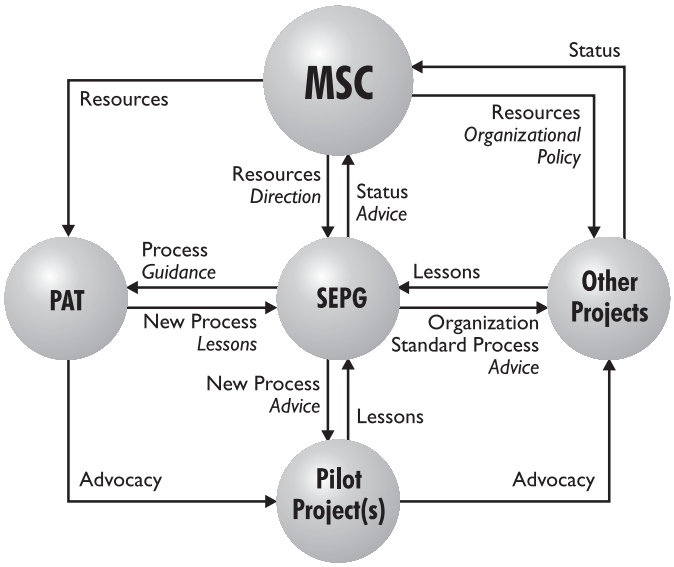


Figure 1. *Process improvement team relationships.*

Using a process to define processes, the PAT defines and documents the new process and identifies valuable lessons learned, under the guidance of the SEPG or an appropriate outside group. Once the new process is documented, the emphasis turns to implementation via a pilot project aimed toward eventual adoption throughout the organization, followed by continuous process improvement. Because the group that guides the PAT deals with the specifics of process definition and documentation, what do the members of the PAT bring to the table?

Who Should Be on a PAT?

The intent of the PAT is to codify (define and document) the organization's best practices, then continuously improve the documented process. Therefore, the members of the PAT should come from among the organization's experts, those who

best perform the existing process. For example, a PAT that defines a configuration management (CM) process should consist of three or four of the top CM managers or technologists from the organization who are involved in process improvement. A significant point to keep in mind when selecting PAT members is that they will codify what they know. Carefully assemble a small PAT (three or four members) from your organization's experts. Do not select whomever happens to be available or whomever you "can afford to lose." Selecting the proper team members is the most important determinant of the PAT's success.

By nature of their abilities, experts are in high demand throughout the organization. The "process for PATs" described later in this article employs a key strategy to minimize the time PAT members are away from their primary duties.

PAT members should have no PAT responsibilities outside of working sessions. Process documentation activities that take place between working sessions are performed by the PAT facilitator and administrative staff, not by the PAT members. Therefore, PAT members are only required to bring their knowledge to the table, not their process documentation skills. The PAT facilitator should be available at least half time while facilitating a PAT.

One benefit of having experts on the PAT is credibility. These experts are the people who know, use, and refine the current processes every day. Because they represent the top performers in the organization, it is their processes that management should want to codify. They also significantly improve buy-in from everyone else in the organization, without which the best processes will likely fail to be adopted by an organization.

Experts also determine the degree of improvement that should be incorporated into initial process definition and documentation. Our experience at pragma Systems Corporation indicates that process definition and documentation is most effective when there is a mix of *as-is* processes and *should-be* processes in the documented process. Experts from throughout the organization have the experience with the current process to ensure that *as-is* and best practices are incorporated into the process. In addition, experts also are sensitive to how much *should-be* process activity the organization can tolerate from a change management perspective.

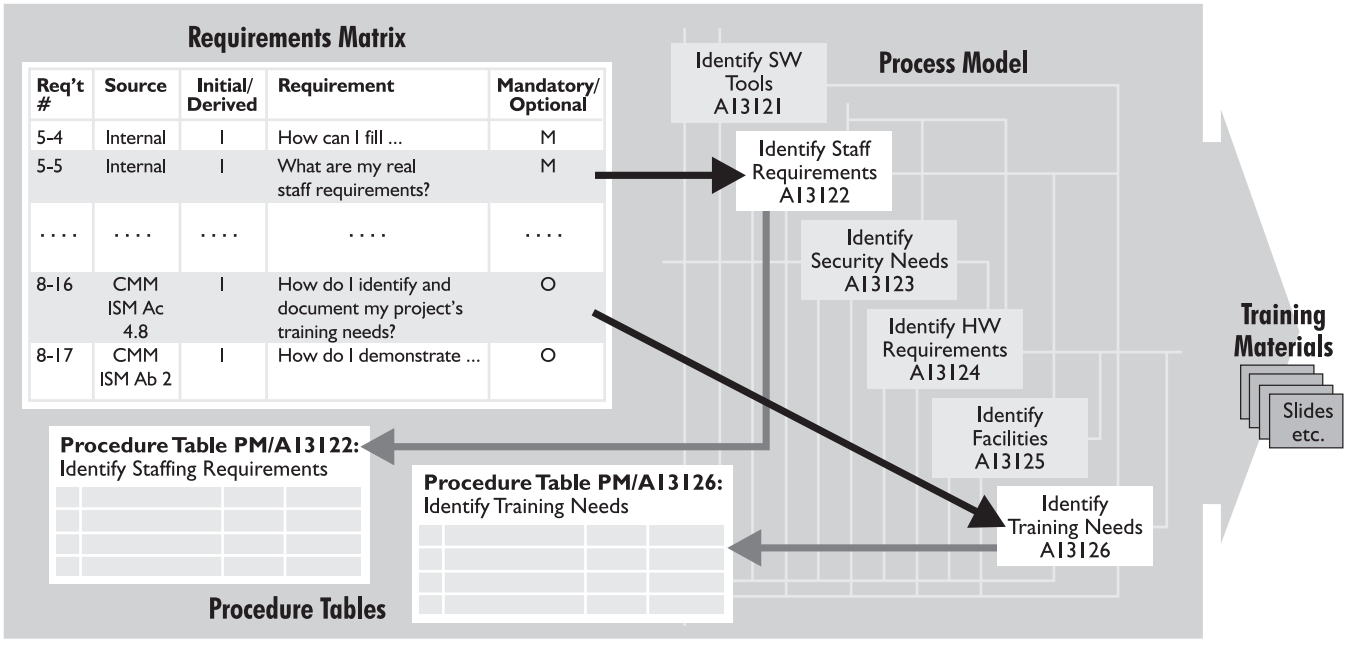
A Process for PATs

Before Forming a PAT

Senior management must make several critical decisions before forming any PATs. These decisions serve as the foundation for all the PATs.

- Decide what artifacts must exist to define and document a process; otherwise, the PATs are left on their own to determine what constitutes a documented process. Several PATs could go off in different directions, documenting processes using different methods and tools, which would result in a morass of inconsistent process artifacts.
- Decide which process(es) should be defined and documented first.
- Decide on the purpose, scope, and viewpoint for each process; otherwise, each PAT will again be left to its own discretion and may or may not hit the mark on the purpose of the process. The PAT may include unnecessary process activities or exclude important activities.
- Decide who will facilitate the PAT (usually an SEPG member or an outside group with the necessary expertise). If chosen by default, the

Figure 2. *Process artifact relationships.*



SEPG may quickly become overwhelmed with facilitating duties.

Once these decisions are made, senior management must follow up by providing the resources required to implement the decisions. The most important resources are the time of the organization's experts who will serve as PAT members and the PAT facilitator.

Senior management must provide funding or other resources to bring in an outside group or individual if the SEPG does not have the expertise to facilitate a PAT through the activities in a process to develop processes. This outside group or individual will train and coach the SEPG members or facilitate several PATs or both. Once these issues are addressed, PATs can be formed with the assurance that they will produce similar and consistent process artifacts that hit the mark relative to the purpose and scope of each process.

The PAT Kick-off Meeting

The purpose of the PAT kick-off meeting, chaired by the PAT facilitator, is to spend a day orienting the PAT team to the task ahead. If the PAT is to define a process that must be compliant with the Capability Maturity Model (CMM) for software, some time should be used to become familiar with the CMM and its structure and the key process areas the PAT must address. And if the process is CMM-related or is to be CMM-compliant, each PAT member must have a copy of the CMM. The PAT also should learn the process that the PAT will use to define and document their process.

In the absence of a separate PAT kick-off meeting, the first day of the initial PAT working session should be used for the same purpose.

Gathering Process Requirements

Just as with development of a software system, requirements serve as the foundation for the process to be defined. Requirements for the process are gathered via a half-day customer focus group meeting and are extracted from any applicable process standards such as the CMM. The customer focus group meeting is extremely important because it involves the future users of the process the PAT will produce. The customers of the PAT go through several brainstorming exercises to identify the internal requirements for the process. Some of the internal requirements will overlap the requirements extracted from a standard like the CMM; however, some requirements will be unique to the organization and must be captured. Ensuring that these unique requirements are captured and incorporated into the process being developed dramatically aids process adoption. This occurs because the customers can see that the process will specifically address their requirements. The PAT facilitator conducts the customer focus group meeting with the PAT members observing. The facilitator also extracts the requirements from any applicable standards, then enters the requirements into a tracking mechanism such as a requirements matrix.

This activity should not be bypassed for two important reasons. First, the requirements are used by the PAT facilitator to guide the PAT through the initial phases of the process model creation. The requirements are the foundation of the process. Second, if compliance with the CMM or some other standard is important, such compliance cannot be demonstrated unless the requirements have been recorded and subsequently traced to the process. Demonstrating compliance later to an evaluator or assessment team would be much more difficult and open to subjectivity on the part of the evaluator or assessment team.

PAT Working Sessions

The first working session of the PAT should occur about one and one-half to two weeks after the customer focus group meeting. The spacing between subsequent working sessions should also be about two weeks. Each working session of the PAT should not exceed three days and should be no less than one and one-half days. To develop the required process artifacts, the facilitator should lead the PAT through a series of highly structured and focused activities. Figure 2 illustrates the relationships between several process artifacts.

Similar to the internal requirements, external requirements are extracted from applicable standards, such as the CMM, and are also entered into the matrix by the PAT facilitator.

All the requirements are translated into activities, inputs, outputs, standards, and resources in a process model. The Integrated Definition method is recommended; however, a number of other graphical process modeling methods may be used. The process model provides a graphical representation of the structure and flow of the process. The process model is not detailed enough to execute, however, unless the process activities are decomposed to the point that the visual model is useless because of extreme complexity. Therefore, the facilitator stops the PAT's process modeling activities when the process model reaches the point where each individual, or lowest-level, activity box in the process model can be described in roughly six to 10 steps in a procedure table. The procedure tables contain the detail necessary to execute the process. PAT members provide the detailed steps for each procedure table to be documented by the PAT facilitator.

By necessity, the process model must have short, clear labels to aid in model readability. To capture the more detailed meaning behind those labels, a process glossary is created to define all the process model terms.

Any time after the process model and procedure tables are created, the PAT spends some time during each working session to map the process requirements to the process model and add this information to the requirements matrix.

When a process is executed, it is helpful to have forms to fill out or examples to follow of products that must be produced. In most organizations, many of these forms and examples already exist. In the spirit of capturing best practices within the organization, the best of these forms and examples are referred to in both the process model and the procedure tables and are physically or electronically collected into a single reference location. As subject matter experts, the members of the PAT collect the forms and ex-

amples, which are then maintained as part of the process by the SEPG or the SEPG's equivalent.

To ensure they develop processes for typical projects within the organization, each PAT is provided a description of the purpose, scope, and viewpoint for the process when they begin. As a result, the PAT creates a process that is somewhat generic or standard with high-level tailoring guidelines that help "real life" projects determine which activities can be eliminated or modified and under what conditions.

Training provides a tremendous boost to any effort to adopt a new process. Therefore, the final set of process artifacts produced by the PAT are training materials that generally consist of overhead slides and instructor notes.

The number of working sessions that each PAT needs varies depending on the complexity of the process and on how much time the MSC wishes to spend on a process. pragma Systems' past experience facilitating client PATs indicates that the minimum number of three-day working sessions should be four, with nine sessions as the upper limit to ensure timely results. The time between sessions should be 10 working days to allow sufficient time to generate the process artifacts for use at the next working session. As a guideline, a four-session PAT should take about three months, and a nine-session PAT should take about eight months.

Customer Focus Group Follow-up Session(s)

If a PAT lasts longer than three months, a follow-up meeting with the customer focus group is recommended shortly after the PAT dissolves and again at two to three months thereafter. These follow-up meetings help with the eventual adoption of the process. Keeping people, especially those who helped generate requirements, informed about the status of process development is critical.

Peer Review

The final meeting of the PAT is an all-day peer review that involves one or more PAT members and two or three peers from within the software organization. Copies of the process artifacts are distributed to the reviewers several days before the review. The process artifacts are then reviewed during the meeting to assess their readiness for pilot testing—not to assess their level of perfection. This distinction is important, since many engineers tend to fall into the trap of trying to engineer as perfect a process as possible before letting others see it, which results in a PAT that spends much more time than necessary to produce a process. Moreover, the process may eventually be revised to account for an issue the PAT did not consider.

Move the developed process artifacts into pilot testing as soon as possible, then make improvements based on actual lessons learned. Following the peer review meeting, the process artifacts are baselined as Version 1.0 and are available for rollout to the organization, usually via a pilot project approach.

Results

The process described above has been used with excellent results by pragma Systems Corporation to define and document several processes. The longest PAT to date defined and documented a project management process. The PAT required seven three-day working sessions over a period of six months and addressed three key process areas of the CMM. The shortest PAT to date defined and documented a process to roll out processes into an organization and improve them via lessons learned and process improvement suggestions. Four two-day working sessions over a period of three months were required.

Intangible benefits include enthusiastic PAT members due to structured and focused PAT activities that lead to a well-defined conclusion, satisfied senior management due to visible and significant progress by the PATs toward well-defined goals, and contagious enthusiasm for process improvement that spreads from the PAT members to the rest of the organization. This enthusiasm often inspires the staff to begin early and use informal or draft versions of process artifacts produced by the PAT.

Summary

Serving as a member of a PAT can be a highly rewarding shining star experience. Successful PATs are small, three to four people, and staffed with the organization's experts. They also exist in an environment where the roles and responsibilities of the various groups involved in process improvement are well understood. A highly skilled PAT facilitator and a well-defined PAT process ensures that the PAT, through highly focused and structured activities, creates the process artifacts identified by the senior leaders of the organization. If these conditions exist, a PAT more quickly defines and documents a process that can successfully enter the cycle of continuous process improvement. ◆

About the Author

Doug Orville has been a senior consultant with pragma Systems Corporation since 1995, where he leads process improvement executive planning workshops, consults with on-going MSCs and SEPGs, facilitates PATs, and teaches customized process improvement courses and workshops. He spent almost 17 years in the U.S. Air Force in various software management and process improvement positions. His assignments included design, development, and maintenance of software for near real-time missile warning, command and control, and intelligence systems; planning, design, and implementation of a major mainframe to client-server migration project; development of a successful software metrics definition process; and leadership of an SEPG for a software organization of roughly 1,200 people.

Orville has a master's degree in business administration from the University of North Dakota and a bachelor's degree in computer science from the University of Wyoming.

8704 Lee Highway, Suite 303
Fairfax, VA 22031-2141
Voice: 703-560-4669
Fax: 703-849-8839
E-mail: dorville@pragmasystems.com

If You Get Straight A's, You Must Be Intelligent

Respecting the Intent of the Capability Maturity Model

David C. Klein
Lockheed Martin Federal Systems

Too often, organizations get caught up in the game of trying to achieve a particular Capability Maturity Model (CMM) rating and do not respect either the intent of the CMM or the positive aspects of a well-planned process improvement program. This article discusses the drawbacks of such a process improvement approach.

Like associating grades with intelligence, too many organizations automatically equate the Capability Maturity Model (CMM) level of an organization with its ability to produce high-quality software. As those involved in software process improvement can attest, this assumption is often far from true. In reality, organizations considered immature can and do produce high-quality software just as mature organizations can and do produce low-quality software. Organizational maturity is no guarantee of success; it merely increases the likelihood of success.

Before the feathers of software quality assurance get too ruffled, let me define my view of quality software. From a project manager's perspective, quality software meets or exceeds all user requirements, is developed within both cost and schedule, and is built to be effectively maintained. I am familiar with projects that have been praised by their customer and user community, have satisfied my definition of software quality, and continue to struggle to achieve CMM Level 2. If we merely assume that a low CMM rating equals low-quality software and that a high CMM rating equals high-quality software, the aforementioned organization's software would be suspect regardless of user or customer satisfaction. Is it better to modify software development processes merely to satisfy various CMM key process areas (KPAs) rather than to improve the software development process? That would be comparable to a student who is more interested in getting a good grade than in acquiring knowledge.

What Is the Problem?

The problem is that the intent of the CMM has been lost in the implementation. The CMM was designed to help organizations improve their software development capabilities and was not intended as some type of awards program or measuring stick. The more process improvement programs (PIPs)in which I participate, the more I am convinced that this misuse of the CMM is more about selling a capability than improving a capability. As long as an organization's Software Capability Evaluation (SCE) rating is good, the process used to acquire the rating is validated. For the same reason, the processes created as part of the PIP are assumed to be valid as well.

Why Is This a Problem?

Is this shift in an organization's process improvement focus a serious problem? Emphatically, yes! A software PIP does not come without a cost. From a software engineering viewpoint, the cost of a well-managed PIP has a justifiable return on investment (ROI) if the PIP increases the quality of the software product being produced. Conversely, if the goal of the PIP is merely to achieve a good SCE rating, the cost of the program is unnecessary overhead and will detract from the overall quality of the software.

Indicators That the SCE Rating Is Valid

From participation in various software process improvement activities, I have identified several issues (explained in the following sections) that can and do influence the validity of the SCE rating. Workers are highly sensitive to perturbations that directly affect their work environment. Therefore, a PIP must be a planned activity that involves the entire organization. PIP leaders and the rest of the organization can only bring about effective process improvement through open communication and diligent training. Otherwise, both apathy and cynicism will create an environment in which a positive ROI can never be realized.

I sincerely believe in the merits of the CMM and believe that there are definite benefits to a well-managed PIP. Furthermore, when software personnel are introduced to the CMM, the merits of the model become self evident.

The bottom line is that many PIP activities succeed or fail based on the level of buy-in from the people involved. It also is important to remember that process improvement is a journey, not a destination.

Proper Staffing of Process Improvement Teams

A well-staffed process improvement team is vital to initiate change within an organization. A successful process improvement team guides, trains, and assists each project within the organization and tracks the overall organizational improvement. Conversely, a process improvement team that lacks the authority to implement changes and is not staffed with the best individuals the organization has to offer is doomed to fail.

Another pitfall to avoid is to staff a process improvement team with senior personnel who mandate change rather than stimulate cooperative change from

the bottom up. Additionally, a good process improvement team should not operate outside the mainstream projects directly affected by the PIP. Care should be taken that the process improvement teams are not divorced from the intimate problems and concerns of the project. A well-managed process improvement team should garner complete, genuine, and uncoerced support and buy-in from each project affected.

Processes-Based Project or Organizational Value

Creating and documenting processes based strictly on satisfying KPA requirements can negatively impact PIP activities. When project personnel are tasked to do activities with little or no visible worth, it detracts from the overall organizational value and productivity. If a KPA for a project is identified as noncompliant, the worth of the KPA should be discussed with the project personnel and a cost-effective solution to satisfy the area should be derived. However, to merely develop and document processes without regard to process improvement or software products is also a waste of time and resources.

Quantify the ROI of Process Improvement

There is overwhelming evidence that CMM compliance has an associated positive ROI. However, the ROI will vary based on a multitude of factors unique to each organization. An organization working toward CMM compliance should attempt to quantify what its ROI should be. In essence, the ROI becomes the fundamental reason for CMM implementation. An ROI will provide the project information that shows how process improvement saves the organization time, money, and resources. Project managers need to know the ROI prior to committing resources to further PIP activities.

Use External Experts for the Right Reasons

External experts can be a valuable resource in a well-managed PIP; however, they should be brought in to aid process improvement rather than to determine how to circumvent a deficient KPA. In addition, external experts should not coach project members in CMM evaluator responses. This is a detriment to genuine process improvement and can quickly create cynicism within a project.

Define Process Improvement in Statement of Work

Rather than merely define an arbitrary CMM level, the government must precisely define what they expect to see as part of a contract organization's software development process. To define a CMM level as a goal can still be done but the definition should detail specific expectations. Therefore, the focus is on mature software processes without regard to a previous or future CMM level. In addition, the government customer is provided the opportunity to show that they are intimate with the CMM and its KPAs. Detailed process improvement requirements can also be used to determine the relative ROI when the statement of work paragraphs are rated during contract performance periods. In this way, the government can give credit for improvement even though the specific CMM goal has not yet been achieved.

Conclusion

Fundamentally, if an organization does not believe in the merits of process improvement, the PIP activities become self defeating. This further erodes confidence in PIP activities, and may culminate in complete dismissal of further attempts to improve the organization's business practices. ◆

About the Author

David C. Klein has worked for Lockheed Martin as a staff software engineer for more than six years. He supports various software test, maintenance, and development projects for the Department of Defense. He has a bachelor's degree in electronic engineering and a master's degree in computer science.

Voice: 719-590-4437
Fax: 719-590-4437
E-mail: David.klein@lmco.com

References

1. Paulk, M., B. Curtis, M. Chrissis, C. Weaver, "Capability Maturity Model (CMM v.1.1)," Technical Report CMU/SEI-93-TR-24, September 1993.
2. Fife, D., B. Brykczynski, D. Heystek, R. Knapper, B. Springsteen, "Conducting Software Capability Evaluations," Institute for Defense Analysis, IDA Paper P-2771, October 1994.

Measurement in Everyday Language

Scott E. Donaldson and Stanley G. Siegel
SAIC

Measurement for measurement's sake is a waste of time and money. Measurements need to be expressed in everyday terms that are familiar to the organization; otherwise, they may be of little value. We present a measurement technique that enables you to measure software products (which can be extended to software systems development processes) in everyday terms familiar—and therefore meaningful—to your organization. This article presents a software product measurement concept that we label product integrity. We illustrate how to measure a software product in terms of attributes and attribute value scales. We briefly describe the application of the measurement approach in the real world.

A developer of software or software-related products wants to stay in business—which is strongly tied to customer satisfaction. Customer satisfaction can be expressed in many ways. First and most important, a product should do what the customer wants it to do. In addition, when a customer pays a developer to develop software products, the customer wants these products to be delivered according to some established schedule and for some established amount of money. In this article, we fold considerations such as these into our concept of "product goodness."

For us, then, product goodness is a multidimensional concept that in this article is called *integrity*. One dictionary definition of *integrity* is "completeness," which can be tied to multiple perspectives represented by product attributes. Often, people think of goodness from one perspective, e.g., manager or developer, or in terms of an attribute, e.g., budget, schedule, or requirements. Our integrity concept allows for blending of multiple perspectives. For example, a manager may think of product goodness as the product being delivered on time or within budget or both. A developer may think of product goodness as the product doing what the customer wants. We think of product goodness as product integrity that folds in all the perspectives.

As shown in Figure 1, we use the notion of displacement (length of a line) in space to derive the idea of a product integrity index that combines all perspectives into a single number. The space of interest is product attribute space; that is, the axes in product integrity space are product attributes. Figure 1 illustrates this notion of displacement for the case of three attributes.

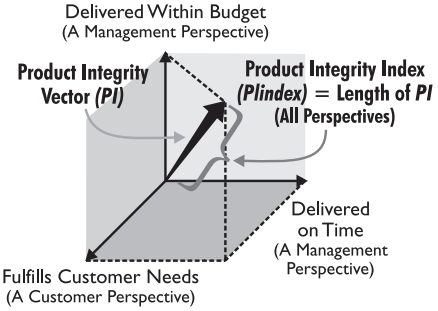


Figure 1. *Object Measurement® uses the notion of a vector length to combine multiple perspectives into a single quantity called a Product Integrity Index.*

We recognize that the attributes we may choose to fold into our notion of "product integrity" would not necessarily be the same as the attributes you would choose. Consequently, the measurement approach we discuss in this article allows you to mold the product integrity concept to the needs of your organization.

This article provides an example set of attributes to define an example product integrity index, a general formula to compute the index, and a summary of how the integrity index concept has been applied in the real world of software systems development.

Measuring Product Integrity

To illustrate the generality of our measurement approach, we now consider a product integrity space that consists of the following five attributes (vs. the three attributes already introduced):

- at_1 *Fulfills specified customer needs.* The product does what it is supposed to do as recorded and agreed to by the customer and the seller.
- at_2 *Can be easily and completely traced through its lifecycle*. The product is "maintainable"—it can be easily updated to incorporate new things, revise existing things, and get rid of things no longer deemed needed by the customer.
- at_3 *Meets specified performance criteria.* How many? How often? How long? These criteria are sometimes considered special cases of the first attribute (at_1).
- at_4 *Meets cost expectations.* The product costs what the customer and the seller agreed.
- at_5 *Meets delivery expectations.* The product is delivered in accordance with schedules agreed to by the customer and the seller in a project plan or updates to the plan.

To understand how we can use the idea of a length of a line in space to quantify the concept of product integrity as a means for measuring customer satisfaction, consider the following five-dimensional vector.

Equation 1. *Five-dimensional measure.*

$$PI = \frac{\begin{bmatrix} at_1 \\ at_2 \\ at_3 \\ at_4 \\ at_5 \end{bmatrix}}{N} = \frac{\begin{bmatrix} CustNeeds \\ Traceable \\ PerfCrit \\ WithinBudget \\ OnTime \end{bmatrix}}{N}$$

The measurement concept presented is a general approach to quantifying almost any object. We call this approach Object Measurement®, which is a registered trademark owned by Scott E. Donaldson and Stanley G. Siegel.

In Equation 1, product integrity (*PI*) is a vector in five-dimensional product integrity space whose components, at_i, are the example product integrity attributes defined earlier. The quantity *N* is a normalization factor that establishes a "product goodness scale." As we subsequently explain, we choose *N* so that the length of *PI* is restricted to the range from zero to one.

To turn Equation 1 into a measurement tool, consider the following questions:

- How can a five-dimensional quantity be converted into a single quantity to simplify measurement interpretation?
- What scales should be established for the attributes?
- What relative weights should be assigned to the attributes?
- How can a scale of values be assigned for the single quantity?

Clearly, there are many sensible ways to address the above questions. The first question deals with simplifying measurement. As Equation 1 indicates, multidimensional expressions of product integrity are possible. Each product attribute dimension contributes to the "length" of the vector *PI*. To convert the five-dimensional quantity in Equation 1 into a single quantity (to represent "quality" or "completeness"), we calculate the "length" of the vector. We call the length of *PI* the Product Integrity Index, or *PIindex*. As subsequently explained, this product integrity vector length, *PIindex*, is simply the square root of the sum of the weighted (w_i) squares of the attributes at_i divided by the normalization factor *N*.

The second question deals with attribute value scales. Many people find it useful and convenient to quantify things in terms of percentages. Thus, a convenient range for an attribute value scale goes from zero to one. Again, for simplicity, we take the approach of limiting the attribute value scales to the range zero to one.

The third question deals with relative weights for product attributes. If we assign the same scale to each attribute (namely, zero to one), we are weighting each attribute equally. For simplicity, we will take this approach. However, you may wish to emphasize one attribute more than the others. For example, if you wanted to give "delivering on time" double the importance of any of the other attributes, you could set its scale to run from zero to two and set the scales of the other attributes to run from zero to one. Equivalently, you can keep all the scales the same and give prominence to selected attributes through the use of weighting factors (w_i).

The fourth question deals with establishing a value scale for the length of *PI*. We select a scale for the magnitude of this vector by choosing a value for the normalization factor *N*. Arguing as we did before, we merely select a scale that ranges from zero to one. For equally weighted attributes, the value of *N* then becomes the square root of the sum of the squares of the maximum values that the attributes at_i can take on. For the case where a product has five attributes each with a maximum value of one, the value of *N* thus becomes the square root of 5.

Defining a PIindex

Based on the four questions introduced above, we can now define a product integrity index, *PIindex*, that ranges from zero to one as shown in Equation 2.

The *PIindex* is normalized to one, i.e., restricted to the range of zero to one. If you want to remove this normalization, remove the denominator.

To illustrate how Equation 2 works, we define value scales for each example software product attribute (at_i). There is a multiplicity of ways such assignments can be made. This example provides insight into ways that you can make such assignments that are relevant to your organization.

Fulfills Customer Needs (at_1)

For at_1, set up a three-value scale based on an Acceptance of Deliverable Form[1] as follows:

at_1 = 1 if the customer returns the form indicating "accepted as delivered."

at_1 = 0.5 if the customer returns the form indicating "accepted with minor changes."

$$PIindex = \frac{\sqrt{\sum_{i=1}^{n} w_i^2 at_i^2}}{\sqrt{\sum_{i=1}^{n} w_i^2 (\text{maximum}[at_i])^2}}$$

where at_i = product integrity attribute
n = number of product integrity attributes
w_i = weighting factor for attribute at_i
maximum $[at_i]$ = maximum value of at_i.

Equation 2. *The PIindex.*

at_1 = 0 if the customer returns the form indicating "changes to be negotiated."

If we wanted to provide more insight into the percentage of requirements fulfilled, we could count such requirements that appear in the product and compare them against some ground truth showing what this number of requirements should be ("shalls" in the language of requirements analysis).

Can Be Easily and Completely Traced Through Its Lifecycle (at_2)

For at_2, the situation can become complicated. Depending on the product, traceability may involve more than the product. For example, if the product is computer code, traceability involves the existence of predecessor products such as design and requirements specifications. If the product is a requirements specification, traceability typically involves documents that a customer may supply such as congressional legislation or corporate policies. More generally, traceability involves such things as product decisions recorded at change control board (CCB)[2] meetings, internal project meetings, and recorded seller and customer management conversations and E-mail between these two parties. To keep things simple, we set up a three-value scale based on the existence of records showing how the product evolved as follows.

at_2 = 0 if nothing other than a customer-prepared statement of work exists that calls for the development of the product.

at_2 = 0.5 if written records exist for some part of the project's lifecycle that show how the product contents are what they are.

at_2 = 1 if detailed written records exist throughout the life of the project showing how the product contents are what they are.

Meets Specified Performance Criteria (at_3)

For at_3, merely set $at_3 = at_1$ since performance criteria are often lumped together with customer needs. If this is not the case in your environment, follow the suggestions offered above for the attribute at_1.

Delivered within Budget (at_4)

For at_4, set up a three-value scale.

at_4 = 1 if the product was delivered for less than the cost specified in the project plan or as modified in CCB minutes.

at_4 = 0.9 if the product was delivered for the cost specified in the project plan or as modified in CCB minutes.

at_4 = 0 if the product was delivered for more than the cost specified in the project plan or as modified in CCB minutes.

Clearly, the above scale places a slight premium on delivering for less than planned cost. The scale also ranks a deliverable delivered for one dollar more than planned cost the same as a deliverable delivered for $3,000 more than planned cost. Again, in your environment you may not wish to place a premium on delivery below cost—but the above gives the idea for how you can establish such premiums (this also applies to the attribute at_5 below).

Delivered on Time (at_5)

For at_5, we set up a scale as follows.

at_5 = 1 if the product was delivered before the delivery date specified in the project plan or before the delivery date as modified in CCB minutes.

at_5 = 0.9 if the product was delivered with no more than a 10 percent schedule slippage. Here, "percent slippage" is calculated by taking the length of time allocated in the project plan for preparing the product or as modified in CCB minutes, dividing that time into the slippage time, and multiplying by 100. For example, if the product was scheduled to be delivered 10 weeks after project start but was delivered 11 weeks after project start, then at_5 = 0.9 because the slippage was (1/10) x 100 = 10 percent.

$at_5 = (1 - X)$, where X is the fraction of schedule slippage as calculated above. For example, if the product was scheduled to be delivered 10 weeks after project start but was delivered 13 weeks after project start, then $at_5 = (1 - 3/10) = 0.7$. For all schedule slippages greater than or equal to the original length of time to produce the deliverable, at_5 = 0 (for example, if a deliverable was to be developed over a 10-week period, any delays greater than or equal to 10 weeks result in at_5 = 0).

The above scale places a slight premium on delivering early. The above scale favors on-time product delivery while allowing for some planning leeway.

Example: *PIindex* Calculation for a Requirements Specification

The product is a requirements specification. After delivery, the customer sent back the Acceptance of Deliverable Form showing "accepted with minor changes." Thus, $at_3 = at_1 = 0.5$. The product was delivered on time, so at_5 = 0.9. The project plan called for 300 hours to be expended on the task to produce the document, but only 275 hours were expended. Thus, $at_4 = 1$. Written records consisting of CCB minutes that show decisions underlying the document's content exist for some part of the project. Thus, $at_2 = 0.5$. The *PIindex* for this requirements specification is therefore the following:

$$PIindex = \frac{\sqrt{0.5^2 + 0.5^2 + 0.5^2 + 1^2 + 0.9^2}}{\sqrt{5}} = 0.72$$

Benchmarks

It is easy to measure, for example, our weight. However, the resultant measurement is generally of little value if our objective is to gain or lose weight. We need weight benchmarks to know whether we are underweight or overweight. Similarly, we need benchmarks for *PIindex*. For example, we can use *PIindex* to establish norms for "product quality" or "completeness." As you gain experience with this index, you can establish goals for various types of products, projects, and seller periods of performance. For example, you can establish goals such as the following:

- Legacy system releases for which little or no documentation exists shall have a *PIindex* not less than 0.75.
- Deliverables for projects whose ultimate objective is to produce a new software system shall have a *PIindex* not less than 0.85.

For this reason and others, it takes time to institute a measurement process. Keep the measurement process simple—otherwise, it will die quickly. "Simple" means "easy-to-collect data and easy-to-interpret information resulting from these data."

Application in the Real World

In this section we highlight how the ideas described above have been applied in the real world of software systems development. This discussion may offer you some ideas for instituting a measurement program in your organization if you do not currently have one or for adjusting an existing measurement program. Reference [1] elaborates on these ideas.

For over five years, we have participated in the development and operation of a software systems development center that could be termed a "software factory." We use the notion of a factory here to suggest that the center manufactures, in a consistent fashion, software products for a diverse customer base. Some customers want minor repairs to existing legacy systems, some want major enhancements, while others require an entirely new software system. The center services from 30 to 50 customers. Customer project sizes range from one person to approximately 50. Typically, customers fund projects for a period not exceeding 12 months, and project budgets range from tens of thousands of dollars to several million dollars. The center produces approximately 100 contract deliverables a month. These deliverables span a broad range of software and software-related products and supporting services.

For each deliverable provided to a customer, we calculate the deliverable's product integrity using the following two attributes: at_1 = "fulfills customer needs," and at_2 = "meets delivery expectations." As each deliverable is provided to the customer, the center uses a Customer Acceptance of Deliverable Form to collect, in part, customer feedback on the product integrity attribute "fulfills customer needs," i.e., at_1. The customer fills out the acceptance form by selecting one of the following choices on the form.

- Deliverable is "accepted as delivered" ($at_1 = 1.0$).
- Deliverable is "accepted with minor changes" ($at_1 = 0.5$).
- Deliverable is "not accepted and changes need to be negotiated" ($at_1 = 0.0$).

In addition, as each deliverable wends its way through the center's product development process, it is tracked by a form that we call the *Deliverable Tracking Form*. This form has entries on it tied to the activities that make up the center's product development process. These activities include such things as peer reviews, product assurance support, technical editing (for deliverables that are documents), and project-level and organization-level management reviews and sign-offs. Two pieces of data that are captured on this form are the customer's required deliverable due date and the date the deliverable is provided to the customer. From this data, we can assign a value to the product integrity attribute for meeting delivery expectations as follows:

- Deliverable is "delivered on time" ($at_2 = 1.0$).
- Deliverable is "delivered late" ($at_2 = 0.0$).

The resulting *PIindex* provides insight into the fulfillment of a customer's needs and meeting a customer's delivery expectations. With a 100 deliverables a month, this simple *PIindex* helps point out areas where there may be (we stress, *may be*) potential problems.

To complement a deliverable's *PIindex*, we also calculate an integrity index for the product development process used to produce the deliverable. The process integrity index is an extension of the *PIindex* concept. Again, to keep our measurement activities simple, we use the back of the Deliverable Tracking Form to capture the data we need to calculate the process integrity index. The back of the form contains a set of six value scales that are tied to the product development process activities on the front of the form. The responsible person, e.g., principal author of a product, technical editor, or product assurance reviewer, circles the appropriate value on the value scale. These circled values are then used to calculate a process integrity index for the process used to produce the deliverable.

We store the product integrity data and process integrity data in a centralized data base. This data base is accessed by a tool that helps us answer questions such as the following:

What is the center's average PIindex for the last three months? The answer to this question provides global insight into how well the center is doing with respect to giving its customers what they asked for on time. An average product integrity value close to 1 means that the center is consistently delivering "good" products, where goodness means "on-time delivery" and "products that do what they are supposed to do."

STSC Measurement Assistance

The product integrity index (*PIindex*) can be useful to organizations that already collect and use measurements. However, if an organization does not have a measurement program in place and management merely wants to check a box, the *PIindex* should not be used as a "silver bullet." In this article, the authors did not discuss the importance of unfolding the data that make up the *PIindex* or how to unfold it because of space limitations (see reference [1] for a complete explanation). An organization may want to use the *PIindex* to see an overall trend or use the *PIindex* as a trigger for action, but to use the *PIindex* solely to analyze and interpret data hides information many organizations need to know. It is important that organizations continue to look at their raw data and the extremes in that data for the messages and trends they contain.

If you are just starting a measurement program, begin with simple measures that address your organization's issues and let the program evolve as you learn. Choose measurements that are important to you, that help you reach your immediate goals and address your issues. As Donaldson and Siegel state, the attributes they chose might not necessarily be the attributes you would choose. If you find it difficult to decide what you want from a measurement program, a good start might be the basics of size, effort, cost, schedule, defects, and rework. Once you find a set of measurements that work for you, you can then combine related information to create your composite metric.

If you want help to determine what measurements your organization needs or want help to get a measurement program running, the Software Technology Support Center (STSC) can help you. We can perform a measurement capability evaluation of your organization, then use the data to return information about your measurement strengths and weaknesses and recommend steps for improvement. We can also support you with training and hands-on coaching while you develop your organizational measurement program.

Beth Starrett, STSC
OO-ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
Voice: 801-775-5555 ext. 3059 DSN 775-5555 ext. 3059
Fax: 801-777-8069 DSN 777-8069
E-mail: starretb@software.hill.af.mil

What has been the PIindex trend over the past two months? The answer to this question provides insight into whether the center's products are improving, staying the same, or declining with respect to on-time delivery and content.

Which projects have deliverables with PIindex values less than 0.5? The answers to this question help senior center management turn its attention to projects that may need extra attention.

What is the center's average process integrity index for the last three months? The answer to this question provides global insight into how well the center is following the systems engineering environment documented practices. An average process integrity value close to 1 means that the center is

consistently following stated practices. An average process integrity value close to 0.5 means that the practices are not being done as documented. There are multiple reasons why this may be the case. For example, some of the practices just do not make good sense given the type of products being produced. Consequently, the practices may need to be rethought. Perhaps employees are simply ignoring the practices. Whatever the case, the process integrity index helps to highlight those areas of the product development process that may need to be improved.

What if the average product integrity score is near 0.5 and the average process integrity score is 0.95? The answer to this question may be that the project teams are following the documented practices but are producing products the customers are not accepting.

From a review of the above measurement observations, decision makers, product developers, and others can focus their attention (and potentially, resources) on those activities that may need improvement. The decision might be to take more measurements and review them carefully. Perhaps the software development process needs to be more closely followed, maybe the process needs to be changed, or maybe the management or the product development staff are overcommitted. Regardless, product and process integrity measurements can be expressed in everyday terms to help achieve customer satisfaction.

Summary

This article describes and illustrates an approach for measuring the "goodness" of software-(related) products. For us, "product goodness" is a multidimensional concept. The label we put on this concept is *integrity*. The following five steps capture the essence of our product measurement approach.

- Decide on the questions that you want or need to address, e.g., are we producing "good" products?
- Select the products from your software systems development process that you want to measure, e.g., requirements specification.
- Identify the product attributes that you want to measure, e.g., for a requirements specification, you might identify an attribute as "at_4, meets cost expectations."
- For each identified attribute, e.g., at_4, define a value scale in everyday terms that are familiar to the organization e.g., delivered for more than the cost estimate = 0.0, delivered for cost estimate = 0.9, and delivered for less than cost estimate = 1.0.
- Using Equation 2, calculate the *PIindex* value. For simplicity, use the formula to restrict values between zero and one. Select weighting factors to reflect your perception of the relative importance of your product attributes.

The measurement approach described in this article can be extended to measure almost any object. In particular, we have introduced how it can be used to measure your software systems development process. This extension, as well as a more in-depth discussion of the ideas presented in this article, are given in reference [1]. ◆

About the Authors

Scott Donaldson, a corporate vice president with Science Applications International Corporation (SAIC), has been in the computer field since 1974. He has a broad range of software engineering experience in the public, private, and commercial industries, including the design, development, implementation, technical management, and evaluation of computer systems application. He is the director of an SEPG that helps a 350-person organization achieve SEI Level 3. He is also the deputy program manager for this more than $28 million business. He received a bachelor's degree in operations research from the U.S. Naval Academy and a master's degree in systems management from the University of Southern California.

SAIC
200 North Glebe Road, Suite 300
Arlington, VA 22203
Voice: 703-516-0603
Fax: 703-516-0618

Stanley Siegel, an assistant vice president with SAIC, has been in the computer field since 1970. Since 1976, he has specialized in the area of software product assurance and co-wrote a textbook on the subject that appeared in 1987. He is a co-author of the first textbook on software configuration management. Together with Scott Donaldson, he wrote a book on software process improvement that was published in March 1997. He is a member of an SEPG in an organization working on approximately 40 software projects of various sizes. He holds a doctorate in theoretical nuclear physics from Rutgers University.

SAIC
200 North Glebe Road, Suite 300
Arlington, VA 22203
Voice: 703-516-0608
Fax: 703-516-0618

Reference

1. Donaldson, S.E. and S.G. Siegel, *Cultivating Successful Software Development: A Practitioner's View*, Prentice-Hall PTR, Upper Saddle River, N.J., 1997.

Notes

1. As part of our software systems development process, we use an Acceptance of Deliverable Form to obtain, in part, customer feedback. When the product is delivered to the customer, the customer reviews the delivered product, decides the product's status, signs the form, and returns the form to the seller. For this example, we have assigned discrete values for the three possible customer evaluations.
2. The CCB is a management support tool that provides a forum for discussing management, development, and product assurance activities. Our concept of a CCB extends far beyond the traditional configuration management control board concept. Simply stated, no matter how well the customer articulates what is needed to be done, no matter how well the seller writes a corresponding project plan, and no matter how well the customer and the seller negotiate the final agreement, once the project begins, things start to change. Furthermore, changes persist throughout the project. Therefore, the CCB is a business forum where the customer and the seller can discuss how to deal with this unknown but anticipated change.

It's Time to Register for the Tenth Annual Software Technology Conference

Dana Dovenbarger
Software Technology Conference

The Tenth Annual Software Technology Conference (STC '98) will be held in Salt Lake City, Utah, April 19-23, 1998.

The U.S. Air Force, Army, Navy, Marine Corps, and the Defense Information Systems Agency (DISA), have again joined forces to co-sponsor STC '98, the premier Software Technology Conference in the Department of Defense. Once again, Utah State University Extension is the conference nongovernment co-sponsor.

The government co-sponsors are Lt. Gen. David J. Kelley (DISA), Lt. Gen. William Campbell (U.S. Army), Dr. Helmut Hellwig (U.S. Air Force), Rear Adm. George Wagner (U.S. Navy), and Maj. Gen. Joseph Anderson (U.S. Marine Corps).

The theme for STC '98, "Knowledge Sharing – Global Information Networks," is shaping up to be a transition conference that reflects the convergence of the Defense Department's tactical and nontactical information systems, processes, people, and policy in support of our war fighters. This theme reflects the broader role of software within the domain of knowledge sharing. Going beyond mere transmission of data elements, knowledge sharing identifies the need for contextual exchange of situational awareness within the dispersed, rapid-paced battlespace in which modern U.S., allied, and coalition forces operate. STC '98 showcases more than just a "software technology conference"; it sharpens the focus of the ubiquitous

STC '98 Vendors – Preliminary List

Abacus Technology Corporation
Abelia Corporation
Ada Core Technologies, Inc.
aimware
Anteon Corporation
Aonix
Army Reuse Center
ATA — DocEXPRESS
Attachmate Corporation
AXENT Technologies, Inc.
Battelle
BMC Software, Inc.
Bookstore
Boole & Babbage, Inc.
COGNOS Corporation
Computer Data Systems, Inc.
Cryptological Support Group
Data Focus, Inc.
DDC-I
Defense Automated Printing Service
Defense Information Systems Agency
Distributive Data Systems
EDS
FedSoft Corporation
Galorath/SEER
Government Computer News
Green Hills Software, Inc.
Hewlett-Packard Company
HQ USACECOM, SEC, ISSC
Hughes Aircraft Company
IBM Corporation
Infodata Systems, Inc.
Informix
Institute for Software Process Improvement
Integrated Chipware, Inc.
Integrated System Diagnostics, Inc.
International Function Point Users Group
Intrinsa Corporation
Lockheed Martin
Logicon, Inc.
Lotus Development
Lucent Technologies
McCabe & Associates, Inc.
MCR Federal, Inc.
Microsoft Corporation
National Security Agency
Naval Undersea Warfare Center
New Dimension Software
Novell
NPLACE
OAO Corporation
Objective Interface Systems, Inc.
OO-ALC/Software Engineering Division
Oracle Corporation
PeopleSoft, Inc.
PMS500 TC
Practical Software Measurement
pragma Systems Corporation
Progressive Software Solutions, Inc. (PROSOFT)
Quality Checked Software
Quantitative Software Management, Inc.
Rational Software
Robbins-Gioia, Inc.
SAIC
SAS Institute, Inc.
SIGNAL Corporation
Software Engineering Institute
Software Program Managers Network
SPAWAR SYSCEN San Diego
SQL Software, Inc.
SRA International, Inc.
Software Technology Support Center (STSC)
Sun Microsystems
Software Technology Process & People (STPP)
Thomas & Herbert Consulting LLC
Tivoli Systems
Tofs
TQL Office HQMC
TRI-COR Industries, Inc.
Tri-Pacific Software
TRW, Inc.
TYX Corporation
U.S. Air Force
U.S. Navy
USA AMCOM
USA AMCOM/CIC
USAF AMC CPSS
Utah State University Extension
VERILOG, Inc.
Viasoft, Inc.
Vitech Corporation
Z Microsystems

role of software, information technology, and information warriors as they support our military capabilities.

We anticipate over 3,500 participants from the military services, government agencies, contractors, industry, and academia.

The Opening General Session will be held on Monday afternoon. In addition to the general session, the co-sponsors have agreed to host a Question-and-Answer General Session on Tuesday morning, where they will answer questions from conference participants. Conference attendees are encouraged to turn in their questions to conference management prior to the conference or to the on-site control room on Monday of the conference week. This is a good chance to get answers from senior leaders on important issues.

Over 500 great abstracts were submitted this year by potential speakers, which made the choice of just over 100 speakers extremely difficult. Participants will be pleased with the selection of speakers and topics. Topics will include but are not limited to

- Capability Maturity – Models, Assessments, Evaluation.
- Client/Server.
- Configuration Management.
- Risk Management.
- DoD Software Policies.
- Outsourcing and Privatization.
- Quality Assurance.
- Embedded Software.
- Open Systems and Architecture.
- Software Engineering.
- Global Information Issues.
- Object-Oriented Technology.
- Measure/Metrics.
- Process Improvement.
- Education and Training.
- Internet/Intranet.
- Project Management.
- Software Acquisition.
- Cost Estimation.
- Technology Adoption.
- Security.
- Data Administration.
- Product Line Engineering.
- Year 2000.
- Knowledge-Based Systems.

There will be special tracks presented by the joint services, the Software Program Managers Network, the Software Engineering Institute, and other organizations.

There will also be a special intelligence meeting held in conjunction with the conference on Wednesday. Top Secret clearances will be necessary for this one-day track. Details are in the registration brochure.

Our exhibit area has grown to over 336 vendor booths. At press time, limited space is still available. Registration information is in the registration brochure. For current vendor information, please check the Web site at http://www.stc98.org. A preliminary list of pre-registered vendors (as of Dec. 10, 1997) include those listed on page 29.

Space rental rate is $1,175 per 10-foot square booth. Late registration received after Feb. 17, should space be available, will rent for $1,275 per booth. A copy of the exhibitor registration brochure with a full layout of the exhibition area is available on the Internet at http://www.stc98.org. For more information concerning the exhibition, E-mail a request to exhibits@LSLP1.usu.edu, use fax-on-demand at 435-797-2358, or call 435-797-0047.

Conference management is coordinating Military Air flights to the conference from the San Diego and Washington, D.C. areas. If interested, please contact conference management at 801-777-7411 DSN 777-7411 or E-mail at wadel@software.hill.af.mil.

One of the greatest benefits of STC is that it provides great networking opportunities. Side meetings and Birds-of-a-Feather meetings are already being scheduled. If you are interested in reserving a time for one of these meetings, please call Carie Kessel at 435-797-0089, and she will be glad to schedule a meeting for you.

Hotel guest room reservations are being taken through the Salt Lake Convention and Visitors Bureau (SLCVB). To reserve your hotel guest room, fill out the form in your registration brochure or call fax-on-demand 435-797-2358 for a housing reservation form. As soon as possible, fax it to the SLCVB Housing Bureau at 801-355-0250. Government-rate rooms go quickly. Buses will be provided to help with transportation between the conference center and city center hotels during conference hours. Be sure to read the instructions on the housing form closely.

The conference fee structure for STC '98 is

Discounted registration fee paid by March 30, 1998	
Active Duty Military/Government	$465*
Business/Industry/Other	$585
Regular registration fee paid after March 30, 1998	
Active Duty Military/Government	$515*
Business/Industry/Other	$635

* Military rank (active duty) or government GS rating or equivalent is required to qualify for this rate.

The official STC '98 registration brochure was mailed in early January. We have made it easier to register early this year. Send in your registration forms with your credit card number *now*, and it will not be charged until March 30, 1998. You no longer have to wait until April to register.

If this issue of *CROSSTALK* was mailed to you, you are on our mailing list. If you had to borrow a copy, please contact us to be added to our mailing list.

You may use our Web site at http://www.stc98.org for further information about STC '98.

If we can be of further assistance, please call or E-mail. This is one conference that you do not want to miss. We will see you in April! ◆

Dana Dovenbarger, Conference Manager
Lynne Wade, Assistant Conference Manager
Software Technology Support Center
OO-ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
Voice: 801-777-7411 DSN 777-7411
Fax: 801-775-4932 DSN 775-4932
E-mail: dovenbar@oodis01.hill.af.mil
wadel@software.hill.af.mil

CrossTALK

Sponsor	***Lt. Col. Joe Jarzombek*** 801-777-2435 DSN 777-2435 jarzombj@software.hill.af.mil
Publisher	***Reuel S. Alder*** 801-777-2550 DSN 777-2550 alderr@software.hill.af.mil
Managing Editor	***Tracy Stauder*** 801-777-9239 DSN 777-9239 staudert@software.hill.af.mil
Senior Editor	***Sandi Gaskin*** 801-777-9722 DSN 777-9722 gaskins@software.hill.af.mil
Graphics and Design	***Kent Hepworth*** 801-775-5555 ext. 3027 hepwortk@software.hill.af.mil
Associate Editor	***Lorin J. May*** 801-775-5555 ext. 3026 mayl@software.hill.af.mil
Editorial Assistant	***Bonnie May*** 801-775-5555 ext. 3023 mayb@software.hill.af.mil
Features Coordinator	***Heather Winward*** 801-775-5555 ext. 3028 winwardh@software.hill.af.mil
Customer Service	***Barbara McDonald*** 801-777-8045 DSN 777-8045 mcdonalb@software.hill.af.mil
Fax	801-777-8069 DSN: 777-8069
STSC On-Line	http://www.stsc.hill.af.mil
***CROSSTALK* On-Line**	http://www.stsc.hill.af.mil Crosstalk/crostalk.html
ESIP On-Line	http://www.esip.hill.af.mil

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

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

E-mail: custserv@software.hill.af.mil
Voice: 801-777-8045, DSN 777-8045
Fax: 801-777-8069, DSN 777-8069

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

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

Reprints and Permissions: Most material in *CROSSTALK* may be reprinted at no charge. Coordinate reprint requests with *CROSSTALK*.

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

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

STSC On-Line Services: STSC On-Line Services can be reached on the Internet. World Wide Web access is at http://www.stsc.hill.af.mil. The STSC maintains a Gopher server at gopher://gopher.stsc.hill.af.mil/. Its ftp site may be reached at ftp://ftp.stsc.hill.af.mil. The Lynx browser or gopher server can also be reached using telnet at bbs.stsc.hill.af.mil or by modem at 801-774-6509 or DSN 775-3602. Call 801-777-7989 or DSN 777-7989 for assistance, or E-mail to portr@software.hill.af.mil.

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

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

Your Effectiveness – A Quiz

Gauge your on-the-job effectiveness by checking the answer that *best* describes your habits, circumstances, and interactions at work.

1. Nothing primes me for work each morning better than

__A. a brisk five-mile jog followed by a hearty, nutritious breakfast.

__B. ingesting so much caffeine I can jump-start dead car batteries with my hands.

__C. a 10:05 a.m., expletive-laden wake-up call from a superior.

2. The first thing I do when I arrive at the office is

__A. review my weekly master schedule and prioritize the day's activities.

__B. tell everyone the story about the guy who parked his car in a bad part of town with two Dallas Cowboys tickets on the seat. (When he came back his window was broken, his stereo was missing, and there were two more tickets on the seat.)

__C. read my E-mail, first deleting any message with a subject heading that contains the words "urgent," "deadline," "meeting," "project," "work," "complaint," "disciplinary hearing," or any message from my boss.

3. For me, a workday typically consists of

__A. three or four cycles of design, coding, and unit testing interspersed with brief, effective peer reviews.

__B. (in descending order of volume) coma-inducing meetings, gripe sessions about what was said during the meetings, general organizational/managerial criticism, distractions, lunch, bathroom breaks, trips to vending machine, actual work.

__C. The Dilbert Zone.

4. One week before an important deadline, you will usually find me

__A. at a party, receiving a bonus check for finishing ahead of schedule.

__B. working frantically to ensure that everything will be ready, i.e., a scapegoat has been chosen, excuses are coordinated, etc.

__C. at the shredder, destroying evidence.

5. When working on a team, I usually

__A. am looked to for advice and am persuaded by other team members to be the leader, after which I conduct brief, yet highly effective working sessions.

__B. gripe that no one will listen to my advice, and with the help of others, manage to turn at least one tiny issue per meeting into a drawn-out power struggle.

__C. N/A. I was barred from human interaction following the bean dip incident.

6. When a colleague criticizes my work, my first reaction is to

__A. take it at face value, then make a conscious effort to improve. No hard feelings.

__B. pretend to care, then call him "Weasel Lips" under my breath.

__C. listen in stone silence, then carve a rebuttal on his desk with a screwdriver.

7. When someone starts to make a request that I consider unreasonable, I

__A. listen quietly, then calmly present my concerns and an alternate approach.

__B. titter nervously while listening, obviously choking back a bitter retort, then break out into a guffaw when they finish speaking; I then feign surprise at their grim expression and exclaim, "You mean you weren't telling a joke?"

__C. N/A—hasn't happened for a long time (see question 6, answer C).

8. When championing an idea that I care deeply about, I

__A. calmly, openly try to persuade others as well as I can, then live with the result.

__B. occasionally try persuasion, but usually resort to massaged data and behind-the scenes maneuvering that would make a campaign manager blush.

__C. usually calm down once the security guards start applying pain holds.

HOW TO SCORE: Count two points per A, one point per B, zero points per C.

13-16 points: Great! You are anybody's dream worker. But verify your answers with your co-workers—some people score high only because they are delusional goobers.

5-12 points: Congratulations: you're an engineer!

0-4 points: Pay no attention to some arbitrary scale that can't account for intangible assets, such as your ability to draw more pay per unit of completed work than any of your colleagues. Emphasize this point on your résumé, which I'm sure you will be updating soon. There's probably something about it in your E-mail.

—Lorin May

Got an idea for BACKTALK? Send an E-mail to mayl@software.hill.af.mil