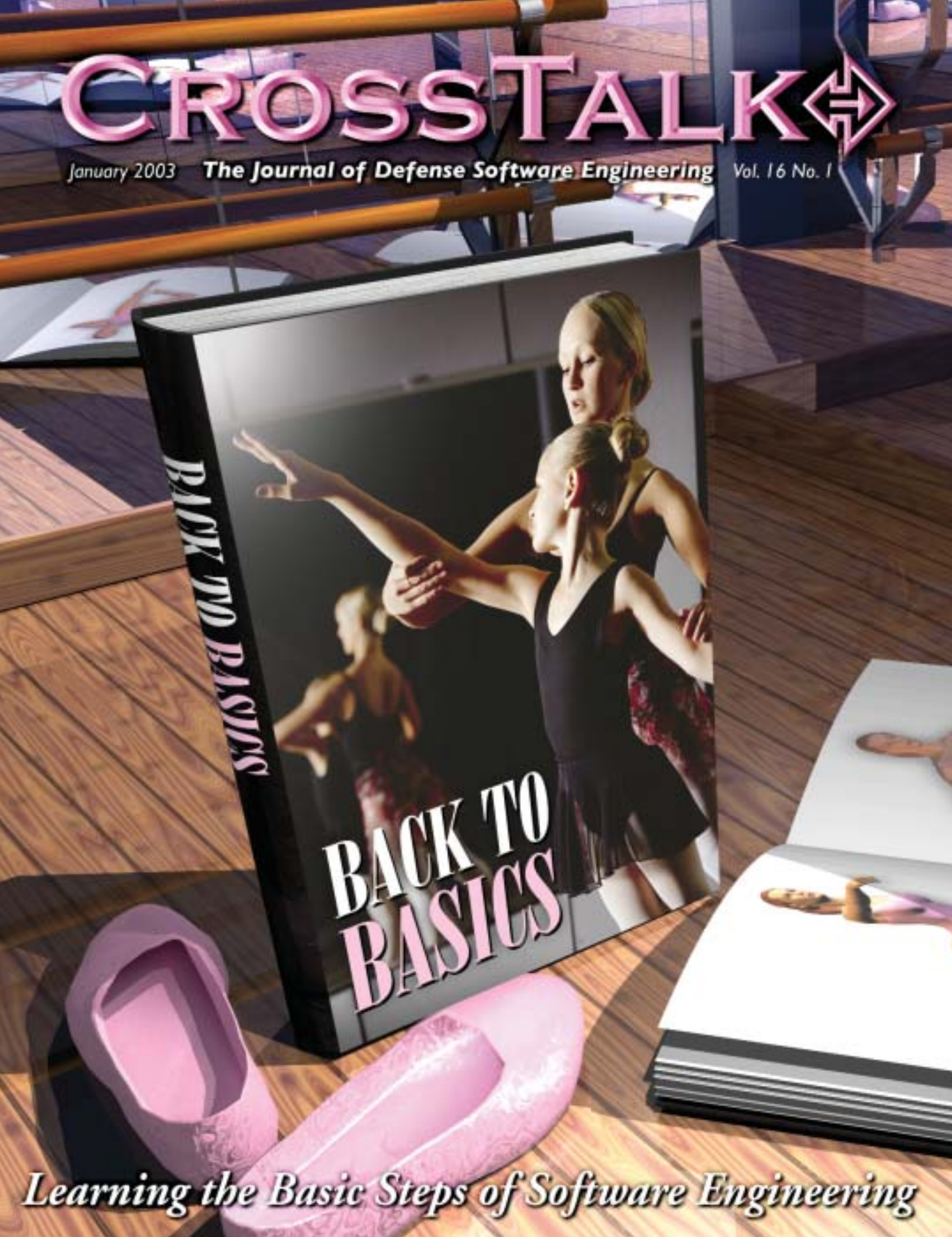

CROSSTALK
January 2003 The Journal of Defense Software Engineering Vol. 16 No. 1
BACK TO BASICS
BACK TO BASICS
Learning the Basic Steps of Software Engineering

Back to Basics

ON THE COVER
Cover Design by Kent Bingham.

Software Engineering Technology

Online Article

Departments

CrossTalk

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

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 23.

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

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

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

Trademarks and Endorsements: This DoD journal is an authorized publication for members of the Department of Defense. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the government, the Department of Defense, or the Software Technology Support Center. All product names referenced in this issue are trademarks of their companies.

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

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

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

The Software Technology Support Center was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

Best Training Includes Going Back to Basics

I recently had the opportunity to attend a meeting conducted by a major program executive officer (PEO). What was the topic? Training. Here were lead engineers and program managers talking about how to train their people in the basics of project management, configuration management, and measurement. It was refreshing! How did they get there? Upon assuming his new duties, the PEO used several definitive references on software development practices: a text, the Software Engineering Institute's "Guide to the Capability Maturity Model® for Software (SW-CMM)," and several training manuals. After some study, he began an initial review of the programs in his portfolio. During this process, he asked questions on key process areas to those managing each program. The message he received? A lack of attention to the basics.

This PEO is not alone. From Ford Motor Co. and the Air Force's Air Logistics Centers to Marvel Comics, all have formal "Back to Basics" programs aimed at reinforcing the organizations' foundation: properly trained people using correct tools, processes, and data. Total Quality Management, Lean Six-Sigma, Benchmarking, ISO, SW-CMM, Capability Maturity Model® Integration℠ – all these depend on having the basics in place. This month's CrossTalk is devoted to some of the fundamental areas necessary to successfully complete software intensive system development.

In *Overview of Project Management*, Tim Perkins, Roald E. Peterson, and Larry Smith of the Software Technology Support Center (STSC) outline the definitions, skills, and processes required for effective project management. Their work is derived from experience in the field and instructing the STSC's Project Management Workshop. Next, in *Delivering Quality Products That Meet Customer Expectations*, Louis S. Wheatcraft emphasizes the importance of establishing a shared vision of the product at the beginning of the project. He then shares best practices and lessons learned in defining product scope such as stating an implementation as a need vs. understanding the customer's need.

Making Measurement Work by Cheryl Jones provides lessons from successful measurement programs and outlines the Practical Software Measurement framework. The current Department of Defense acquisition environment fosters using performance-based contracts with prime contractors for major systems serving as integrators. This makes it crucial to understand critical software architecture, risks, and team capabilities to know what metrics to build into those contracts. A successful measurement process must be a way of doing business and the basis for making fact-based decisions.

Also from the STSC is *But I Only Changed One Line of Code!* by Theron R. Leishman and Dr. David A. Cook. This one-act play provides motivation for the application of one of the most widely accepted software best practices: configuration management. Although generally accepted, basic configuration management activities are often ignored, resulting in serious negative impact on software development and acquisition projects. This article introduces basic software configuration management concepts and the rationale for its implementation.

In *Risk Management Applied to the Reengineering of a Weapon System*, Claude Y. Laporte and Guy Boucher briefly describe a systems engineering process and discuss the application of risk management practices to the reengineering of operator console stations of a missile weapon system, including 12 lessons learned.

Louis A. Poulin's article, *High Quality, Low Cost Software Inspections*, cites work by Ronald A. Radice in his book, "High Quality, Low Cost Software Inspections," and defines inspections, peer reviews, walk-throughs, and structured reviews. He explains that while these are all terms that are used interchangeably in software engineering, the activities are rarely carried out consistently in the course of developing an application.

Are formal methods the answer to providing high-quality software through mathematical rigor? In *Application of Lightweight Formal Methods in Requirements Engineering*, Vinu George and Dr. Rayford Vaughn explain how using formal methods is important to achieving correctness, consistency, and developmental understanding. However, the degree of formalization must be carefully planned. An evolving approach – lightweight formalism – which is less rigorous than normal, can be advantageous.

A wealth of information exists on each of these fundamental topics. The STSC's Web site, <www.stsc.hill.af.mil>, contains additional references. We hope this collection of articles provides you a quick review that you can use when training your team.

Glenn A. Palmer

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

Overview of Project Management

Tim Perkins and Roald E. Peterson
Software Technology Support Center/SAIC

Larry Smith
Software Technology Support Center

Every organization or program creates and implements projects to help it move toward its goals. Every assigned project manager wants to be successful in executing assigned projects, and a number of standard practices exist to assist and guide the project manager. This article is extracted from the Software Technology Support Center's soon-to-be-released condensed guidelines for software acquisition. It outlines some of these key practices that, while are common sense, are not always common practice.

This article provides a brief overview of project management, including its purpose, activities, and responsibilities. The beginning sections discuss what projects are, what project management is, and what project management generally entails. Next is a summary of project life cycles and their phases, along with the processes and activities of project management. The article concludes with checklists, definitions, and further resources. The content of this article has been condensed from multiple sources that are listed at the end along with other recommended Web resources that provide more detail and direction for managing projects. Check them out!

Projects and Programs

A project is a group of activities undertaken to meet one or more specific objectives. These objectives could include solving a problem, building or upgrading a system or product, launching a product or service, implementing a strategic plan, changing a process, or one of many other unique efforts.

Projects can differ in size from small and simple to large and complex. Because they are designed to accomplish specific objectives, projects are temporary and have specific starting and completion dates. Ongoing operations, such as running a maintenance facility or publishing a magazine, are not projects. However, performing a specific aircraft avionics upgrade or printing a monthly issue of a magazine are projects.

Limited, specific performance time frames and objectives are how projects differ from programs. Programs are generally much larger efforts than projects with a longer duration. Relative to projects, they are ongoing rather than temporary efforts.

While this article focuses primarily on project management, most of what is presented will also apply to programs. Programs are made up of multiple projects and in many cases can be treated as longer, more complex projects.

Projects are often divided into smaller components or activities, usually based on technical and functional disciplines such as engineering, manufacturing, testing, and procurement. The relationships among programs, projects, and activities are shown in Figure 1. Some projects are divided along product lines instead of activities.

Projects are successfully completed when their objectives have been achieved. Projects should be terminated when management can see that the projects will fail to meet their objectives.

Project Management

Project management is that discipline that employs skills and knowledge to achieve project goals through various project activities. It involves controlling costs, time, risks, project scope, and quality through project management processes.

Project management includes the following functions:

- Planning. Planning the project and establishing its life cycle.
- Organizing. Organizing resources such as personnel, equipment, materials, facilities, and finances. Coordinating work and resources.
- Leading. Assigning the right people to the right job. Motivating people. Setting the project's course and goals.
- Controlling. Evaluating project progress and, when necessary, applying changes to get it back on track.

Performing these functions in an organized framework of processes is the job of the project manager (PM).

Projects rarely succeed by themselves. They must be planned and executed. Projects must have specific support from management, general support from the organization, and appropriate participation from the customer. To be successful, projects must also have a responsible and empowered manager to drive, direct, and monitor them.

Project Manager

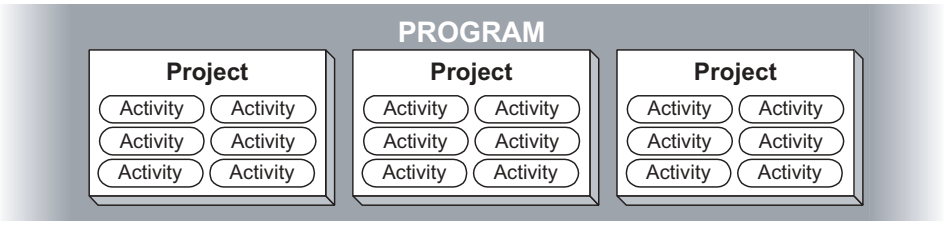
The selection of a PM has a major effect on project success. The PM should have the skills, knowledge, and personality necessary to bring the project to fruition. In addition to these traits, the PM must be given the level of responsibility and authority necessary to perform the job.

The PM's actual role depends on the structure of his/her organization, which can be function-oriented, project-oriented, or some type of matrix in between. In a heavily project-oriented organization, the PM may have relatively unlimited authority, answering only to upper management. At the other end of the spectrum is an organization that manages by function. The PM must deal with functional managers as equals, or possibly even superiors, and negotiate for resources. Most organizations fall somewhere in between these two extremes. Figure 2 depicts the level of PM authority associated with different types of organizations.

It is essential that the PM understands the organization's structure and knows the level of authority that goes with the job. It is also essential that upper management grants authority and establishes an environment that will enable the PM to successfully accomplish the project objectives.

PMs need both management and technical skills. The key management skills are those needed to perform or direct project

Figure 1: *Relationships Between Programs, Projects, and Activities*



management activities, which are listed in Table 1.

The PM's technical skills should include at least some technical understanding of the project field. Remember, however, the PM will not typically be doing the technical work but will be directing the work that others do. The essential level of PM expertise is the ability to understand what others are doing, but not necessarily how they do their jobs.

Process Description

To understand how a project is brought to fruition, the PM must understand project processes. The PM must also understand the differences between program and project life cycles and the difference between life-cycle phases and project-management processes.

Product Life Cycle

As stated earlier, programs are generally large efforts spanning long periods of time and are composed of multiple projects. They are usually associated with developing or acquiring systems such as aircraft, weapons, training operations, communications, etc. An example of a product life cycle with its associated phases and products is shown in Figure 3.

This example has five phases – planning through operation – each producing a specific output. At the end of each life-cycle phase, a decision is made to either continue or not continue to the next phase. When the final product is completed, it is implemented, and after being in operation for some length of time, it is retired.

Various industries employ different life cycles, depending on their products. Each phase of a product life cycle can consist of one or more projects. Note that in the commercial world, a product life cycle is often described as the length of marketability of a product. We must not forget, however, the potentially long and costly maintenance or sustainment phases.

Project Life Cycle

Projects, like products, have life cycles and are usually performed in phases. Each phase accomplishes specific work toward reaching the project goal and produces one or more deliverables. These are tangible, real items used in attaining the final goal of the project, and could include plans, studies, designs, or software or hardware prototypes. The end of a phase is defined by completing its deliverable. Figure 4 (see page 6) illustrates an example of a very generic project life cycle with its phases and major deliverables.

While major aspects of project management are applicable across all projects, life cycles may vary depending on the type of project and the organization performing the work. It is important to implement an appropriate life cycle for the product. This article only deals with a generic project life cycle for a generic product or deliverable.

Project Phases

The phases identified in Figure 4 are common across most projects. However, they may be called by different names or split into additional phases. They may even be iterative where, for example, a prototype is designed, built, and tested, then the results are used to design, build, and test a new prototype. Project phases should, in most cases, be comparable to the generic project phases discussed in the following sections.

Definition Phase

This phase begins when upper management creates a project charter that defines the project's purpose and identifies the PM. The charter should also include a statement of support authorizing the PM to perform his/her functions. During this phase, the project rules are defined. Both the PM and stakeholders determine the project's goals, scope, and constraints. Key individuals and groups are identified as members of the

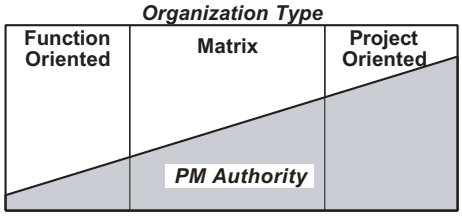
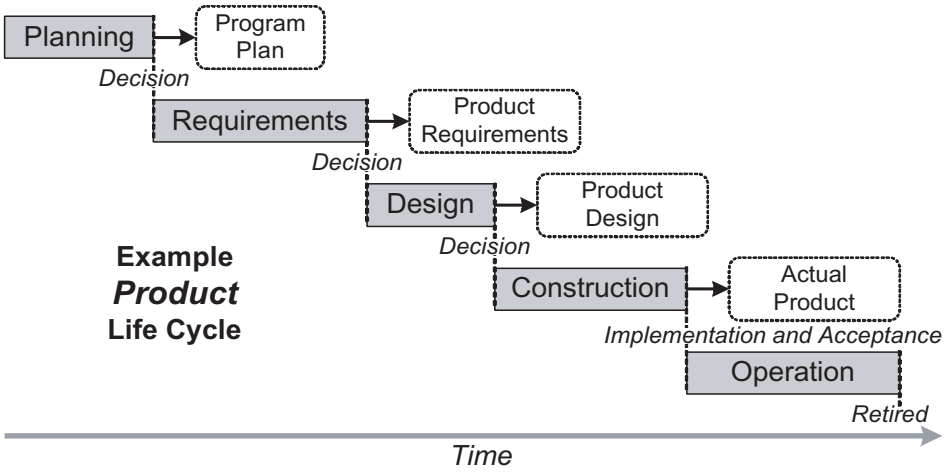


Figure 2: *Organization Type and Project Management Authority*

Table 1: *Management Skills for Project Managers*

Skill	Description
Integration Management	Coordinate project plan development, execute the plan, and manage the change control process to ensure that all aspects of the project are working together.
Scope Management	Establish project scope at the start. Develop and implement plans and procedures to verify that scope is achieved and maintained. Define and oversee the process for controlling changes to the scope.
Risk Management	Identify potential risks. Mitigate large risks and plan how to deal with smaller risks. Monitor the project to detect and resolve problems.
Time and Schedule Management	Estimate the duration of project activities, the proper sequence of these activities, and develop and control the project schedule.
Cost and Budget Management	Estimate project costs and develop and control the project budget.
Quality Management	Establish and control processes to ensure project goals are met to the stakeholders' satisfaction. This includes quality planning, quality assurance, and quality control.
Communications	Define methods and lines of reporting and information distribution: Who gets reports and project information? How often? What is the content?
Procurement Management	Oversee procurement and delivery of materials, equipment, and services needed for the project. This includes planning, solicitation, source selection, and contract administration.
Human Resources Management	Develop good leadership qualities. Plan team organization, obtain the right people to staff the positions, and develop their skills as individuals and as a team.
Earned Value Management	Develop a systematic approach to project control integrating cost and schedule control with performance control.

Figure 3: *Example of a Product Life Cycle*



Note: In some life cycles, e.g., evolutionary or rapid prototype, the output of a design phase may be a product prototype.

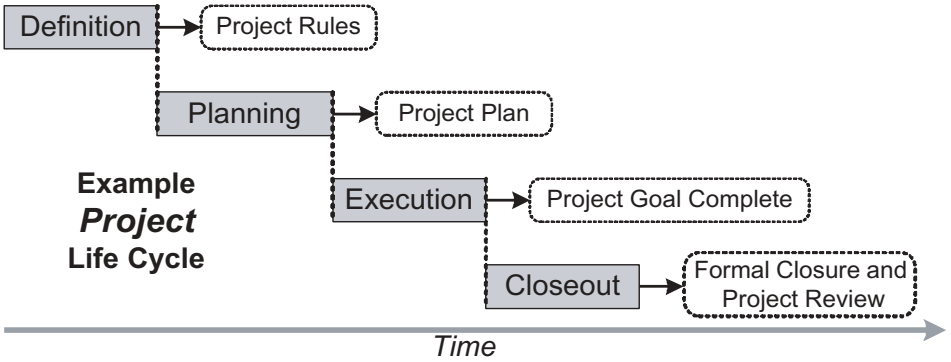


Figure 4: *Example of a Generic Project Life Cycle*

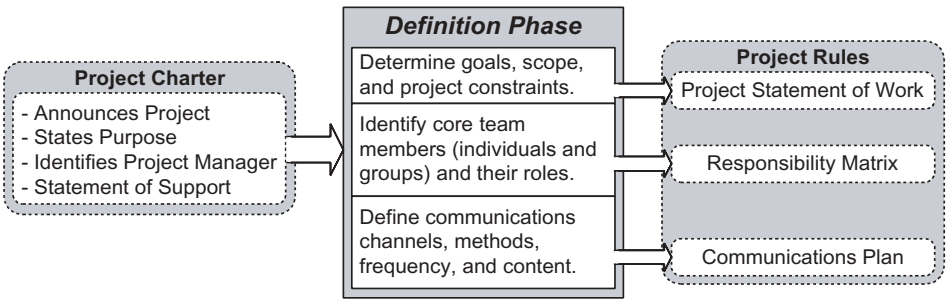


Figure 5: *Project Definition Phase*

project core team, and their roles are defined by both the PM and upper management. Upper management along with the PM also defines communications channels, authority, and the chain of command.

These project rules are written in three documents: the project statement of work (PSOW), the project responsibility matrix, and the project communication plan. The PSOW establishes the scope of the project and documents what is to be accomplished. For an internal project, the PSOW becomes the primary requirements document. However, the PSOW is not the same as a contract statement of work (SOW). For a project where much of the work is contracted, the SOW is a binding, contractual agreement. (See Table 2 for a description of these documents and other terms.) Figure 5 depicts the input, major activities, and products of the definition phase.

Planning Phase

The planning phase uses the project rules as a foundation and defines the path to achieve the project goals. It is performed by the PM and the core project team, which interfaces with appropriate elements of the organization, and identifies the actual work to be done. It includes estimating schedule, cost, and resources required to perform the work, and produces plans to serve as a baseline and direct the work. A key part of schedule planning is identifying the critical path. This is the chain of interdependent, sequential project activities that takes the longest time to complete, and thus determines the minimum schedule for the project. Planning also includes risk identification and risk reduction efforts. The results of the planning phase become the project plan.

Figure 6 shows the inputs, activities, and products of the planning phase. Note the feedback loop from the phase activities to the project rules. This indicates that the rules may need to be modified after more detailed analysis in this phase reveals deficiencies or inefficiencies in the rules. This illustrates the iterative nature of project management. Remember, the project plan is fluid and the PM should expect changes.

Execution Phase

With a project plan for guidance, the actual project work can begin in earnest. This is the phase where project goals are achieved. While Figure 7 may make it look far simpler than the planning phase, the execution phase entails directing the various work groups in their activities, monitoring their progress, solving problems and resolving issues that will certainly come up, making changes to the plan, and coordinating these changes. (These activities are part of the executing and controlling processes discussed in the project processes section.) If your planning has been done well, you will have a smoother ride through this phase. This phase is complete when the product is complete, the project goals are reached, or the project is terminated.

Closeout Phase

The closeout phase begins with the delivery of the product or completion of the project goals or project termination. It consists primarily of tying up loose ends. Any unresolved issues from the contract or SOW are resolved in this phase. The contract is signed off as fulfilled, and all other paperwork is completed.

A very important activity of this phase is assembling the project history. This is a summary of all that has been accomplished. It should include information that will allow either you or a follow-on PM to understand what was done and why. Of particular importance is a compilation of lessons learned from the project so either you or others in your organization can do things better on the next project. Figure 8 summarizes the closeout phase.

Project Processes

The Program Management Institute defines five major process groups used in projects:

Table 2: *Definition of Basic Project Management Terms*

- **Baseline** – A standard against which future status, progress, and changes are compared and measured. Most plans developed during the planning phase are used as baselines. The budget usually serves as one baseline, the schedule as another, etc.
- **Communications Plan** – A document that defines the lines, content, method, and frequency of communications between the project manager, members of the project team, stakeholders, and management.
- **Critical Path** – The sequence or chain of interdependent activities in the project that takes the longest time to complete. This sequence determines the shortest schedule for the project. Any delay in a critical path activity increases the project schedule.
- **Life Cycle** – The complete set of phases something goes through, beginning with its conception and ending with its retirement from service.
- **Process** – A series of related activities or steps that accomplish a specific purpose.
- **Project Charter** – Document that announces the project by name, states its purpose, identifies the project manager, and announces his or her authority.
- **Project Manager (PM)** – Individual with responsibility and authority for directing the project.
- **Project Statement of Work (PSOW)** – Document that defines the goals, scope, and constraints of the project. It states what needs to be done, not how to do it.
- **Responsibility Matrix** – Document that identifies members of the project team and defines their roles.
- **Stakeholders** – Those persons and organizations that have an interest in the performance and completion of the project. The customer or user of a product created through a project is usually a primary stakeholder.
- **Statement of Work (SOW)** – A contractual document that defines the work to be performed for a specific project under contract.
- **Work Breakdown Structure (WBS)** – A breakdown of the project into its constituent's tasks or activities. It lists the specific work needed to complete all aspects of the project.

initiation, planning, executing, controlling, and closing. Processes are sequences of activities that accomplish specific functions necessary to complete or enable some portion of the project. These are not phases themselves but can be found both in projects and in each major phase of a program or large project. Because the activities in later phases may require changes in the products of earlier phases, these processes become iterative and often overlap phases as well as each other. An example would be an issue in the execution phase requiring a change to plans made in the planning phase. This overlap is shown in Figure 9.

Initiation Process

The initiation process consists of formally validating or authorizing the project. It often includes some form of analysis such as a feasibility study, a preliminary requirements study, a concept of operations, or a preliminary plan.

Planning Processes

Planning processes establish the scope or boundaries of the project. They lay the foundation and define an expectation baseline. Future proposed changes are evaluated against this baseline. What must be balanced here and throughout the project are schedule, cost, quality, and scope. Changes to the scope of the project will almost certainly affect at least one of these, requiring changes in the others to achieve balance again. Likewise, changes in one or more of these three constraints will require changes in the others and/or changes to the scope or expectations of the project. This balance is shown in Figure 10. Note that these do not necessarily define the project scope but they do constrain it.

Again, a manager can control any three of these four constraints. For example, if one chooses to control schedule, cost, and quality, the functional capability of the product may have to be reduced. Likewise, if one attempts to expand functional capability (i.e., requirements bloat) while maintaining cost and quality, the schedule may have to be relaxed to maintain the balance.

Other planning processes include the following:

- Define activities needed to perform the project.
- Estimate activity duration.
- Estimate a minimum schedule and develop a project schedule.
- Conduct risk management.
- Develop communications planning.
- Conduct staff planning.
- Develop organization definition.
- Sequence activities.
- Conduct resource planning.

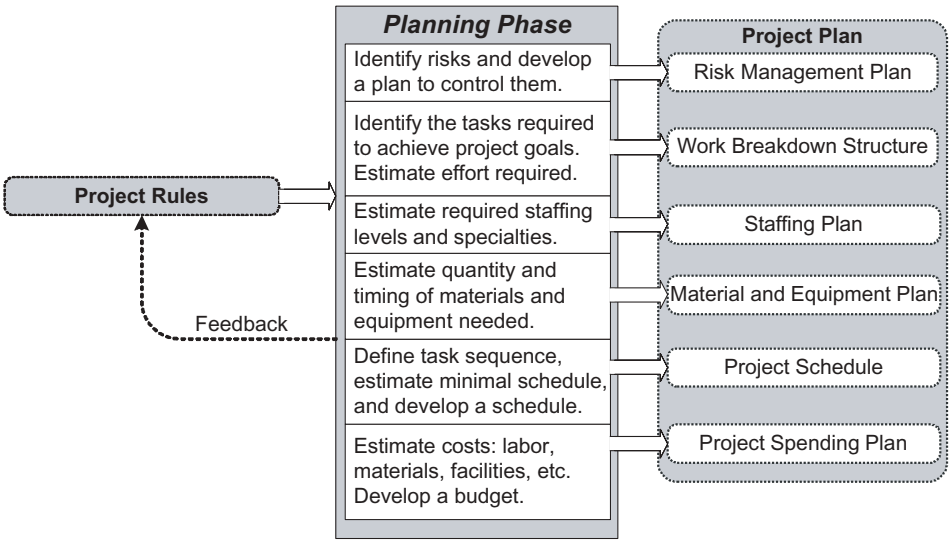


Figure 6: *Project Planning Phase*

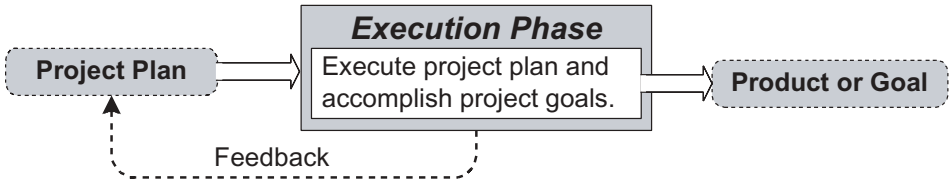


Figure 7: *Project Execution Phase*

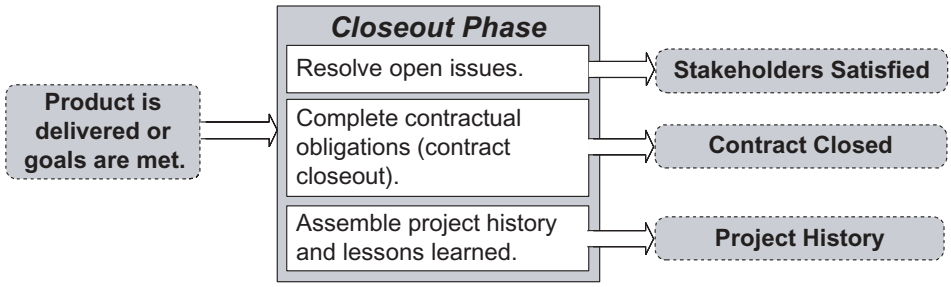


Figure 8: *Project Closeout Phase*

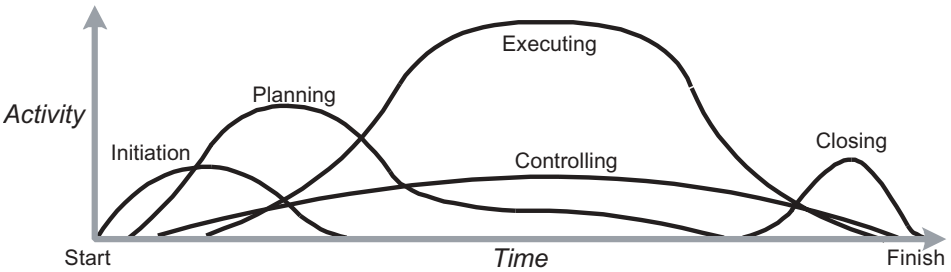


Figure 9: *Project Management Processes Overlap* [1]

- Estimate costs.
- Develop a spending plan or budget.
- Conduct quality planning.
- Conduct procurement planning.
- Develop a project plan.

Executing Processes

The executing processes are those that direct or enable the actual work of the project. They consist of the following:

- Execute the project plan.
- Perform quality assurance activities.
- Perform procurement activities.

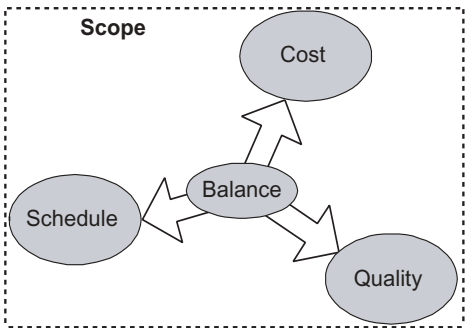


Figure 10: *Balancing Constraints Within Project Scope*

- Develop team and individual competencies.
- Communicate to team members and stakeholders.

Controlling Processes

Controlling processes are ongoing throughout most of the project. They include verifying that the project is proceeding according to plan or determining where and how much a deviation is occurring. They are absolutely essential to the progress and success of the project. They include the following:

- Monitoring, measuring, and reporting the performance of project activities.
- Verifying the project is continuing within scope.
- Controlling changes to the project scope.

These processes are, in turn, enabled by these supporting processes:

- Schedule control.
- Cost control.
- Quality control.
- Functional scope control.
- Risk monitoring and control.

Earned value management techniques, if properly employed, have been shown to be a worthwhile approach to indicate project status, progress, and trends toward successful completion.

Closing Processes

The closing processes are accomplished following the completion of the project objectives. Their purpose is to resolve any open issues, complete any paperwork required for formal completion of the project, and gather information useful for evaluating project performance for future reference. The first process is contract closeout, where any remaining contract issues are settled. The other process is administrative closure, where formal documents terminating the project are generated, and an appropriate history of project performance and lessons learned is gathered.

Project Management Application

Applying the previous information to a real project will depend on several things. A PM assigned to an ongoing project has little control over how the project is set up. In this case, the new PM will need to quickly learn the following:

- Project purpose and objectives.
- Project phases and deliverables.
- Project budget and current spending status.
- Project schedule and current status.
- Current problems and issues.
- Major risks.
- Project team organization and contacts.
- Project management processes in place or planned.
- Life cycle of the product the project is supporting.
- Communications that detail who gets what information and when.

A new project requires the PM to learn or establish the items in the previous list. After understanding the purpose and goals of the project, the PM will need to select an appropriate project life cycle. If it is a small, straightforward software development effort, all the software development life-cycle phases are performed as part of the execution phase, as shown in Figure 11. Remember to distinguish between project phases and software development phases. Also note that this example portrays only one of several possible life-cycle models.

If the software development effort is larger or more complex, the development life cycle will still be performed in the execution phase of the overall project or program. However, each phase of software development now becomes a project in its own right, with all the phases of an individual project. This is shown in Figure 12.

If the PM is managing a project team that is developing software, then he or she will manage both project and software development phases. If managing a contract effort, the PM will manage the overall project, but a contractor PM will manage the actual development effort.

With a contracted software development effort, you will also need to add a procurement phase to your project. This additional phase will take the outputs from the previous stage, including a SOW, and perform procurement activities to contract with an outside organization to perform the work. This additional phase is shown in Figure 13.

Knowing the project goals and selecting a project life cycle establishes the foundation on which to build the project.

Figure 11: *Software Development Phases are Part of the Project Execution Phase*

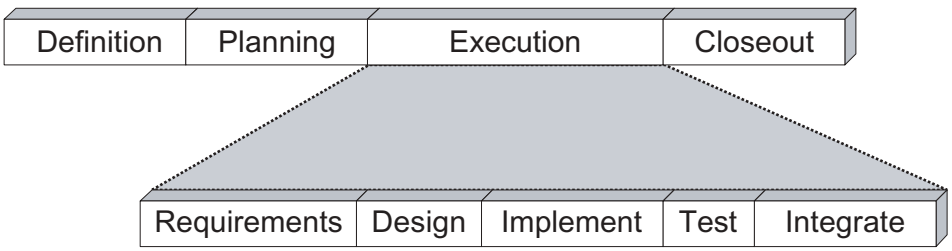
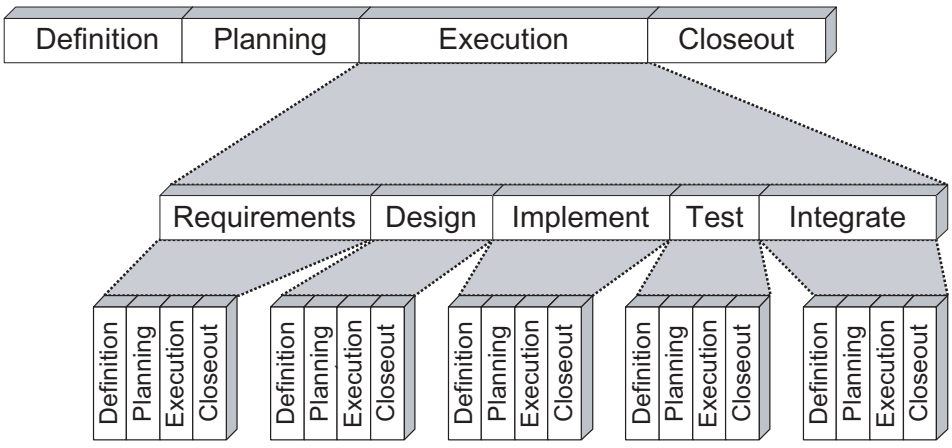


Figure 12: *Complex Development Phases Become Projects Themselves*



Project Management Checklist

This checklist is provided to guide you in essential actions to ensure your project is on track in meeting cost, schedule, and performance requirements. If you cannot check an item off as affirmative, you need to either rectify the situation or develop a contingency plan to solve problems that may arise. For example, if the staff does not have sufficient technical skill to do the work, you will need to remedy the situation either by providing training or by obtaining sufficiently skilled people.

Beginning a Project

- □ The project has specific goals to accomplish, and you understand the reasoning behind them.
- □ All stakeholders (interested parties) understand and agree on the expected project outcomes.
- □ Upper management is solidly behind the project.
- □ You understand the level of authority you have been granted in relation to the project and the rest of the organization, and the level of authority is appropriate.
- □ You understand how the organization

operates, including how to get things done within the organization.
- ☐ You understand what you are responsible for delivering at both a macro and a micro level.
- ☐ You know the high-priority risks your project faces.

During Project Planning

- ☐ You know which external interfaces are not under your control.
- ☐ You know the estimated size of the software to be developed, and how the estimate was made.
- ☐ Funding has been allocated for the project.
- ☐ A credible budget has been prepared, based on project scope and work estimates.
- ☐ Adequate time has been allocated to complete the project.
- ☐ Adequate staff is or will be available to complete project tasks.
- ☐ The project staff has sufficient expertise to perform the work.
- ☐ Facilities and tools are or will be available for the project team.
- ☐ You know of potential funding cuts and when they might come.
- ☐ You know what major problems have plagued projects of this type in the past.
- ☐ An appropriate life cycle has been selected for the project, and you understand that life cycle.
- ☐ You have a credible work breakdown structure.
- ☐ All requirements have work tasks assigned to fulfill them.
- ☐ All work tasks are associated with project requirements or support activities.
- ☐ Special requirements or constraints are documented.
- ☐ You have a budget, schedule, and performance baseline established and documented.
- ☐ You have identified the critical path for the project.
- ☐ You have a process established to monitor the project and detect problems and departures from the baseline.

During Project Execution

- ☐ You know what your project's expenditures are to date and any difference between those and your budget.
- ☐ You know the status of project activity completion along the critical path and any difference between that and the schedule.
- ☐ You are aware of any issues or problems with quality or performance that may impact the critical path.
- ☐ You are aware of any contract performance issues.

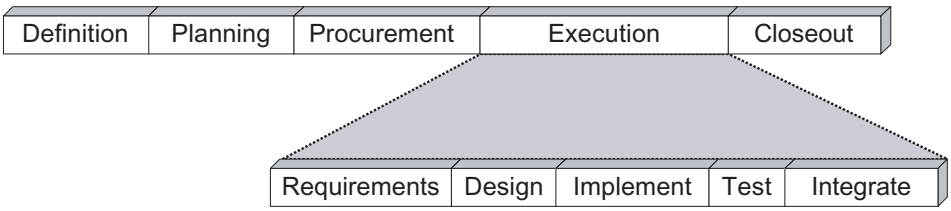


Figure 13: *Project with Procurement Phase*

Conclusion

All project managers desire to bring their projects to a successful conclusion. The typical success factors are meeting cost, schedule, and quality objectives within the allotted scope while also meeting the associated customer expectations. The standard practices to accomplish this desire have been outlined in this article. They include the following: obtaining and maintaining the necessary support from the project sponsor, support organization, and customer; employing an appropriate life cycle and breaking the project into success-oriented phases; setting aside an appropriate amount of time to understand the expectation of key stakeholders; adequate planning to accomplish key objectives; and finally, executing the project plan while balancing the naturally occurring changes with a companion control and tracking system.◆

Reference

1. Project Management Institute. A Guide to the Project Management Body of Knowledge. 2000 Ed. John Wiley & Sons, Inc., 1999.

Resources

1. AllPM. Project Managers Home Page. <www.allpm.com>.
2. Best Manufacturing Practices. TRIMS Risk Management and Best Practices Software downloads: <www.bmpcoe.org/pmws/index.html>.
3. Best Manufacturing Practices Library. Download know-how software and copies of DoD 5000.1, 5000.2, etc. <www.bmpcoe.org/pmws/download/knowhow.html>.
4. Can-Plan Project Management. Software download. <www.geocities.com/billmcmillan2000/CAN-PLAN.html>.
5. Baker, Kim and Sunny. Complete Idiot's Guide to Project Management. 2nd Ed. USA: Alpha Books, 2000.
6. Department of Defense. Defense Acquisition Deskbook. <http://web2.deskbook.osd.mil/default.asp?>.
7. Department of Defense. Defense Acquisition Deskbook, Program Management. <http://web1.deskbook.osd.mil/CS_PM.asp>.
8. Defense Systems Management College. Program Manager Magazine <www.dsmc.dsm.mil>.
9. Department of Defense Software Clearing House. <www.dacs.dtic.mil>.
10. Gantthead Online Community for IT Project Managers. <www.gantthead.com>.
11. Department of Defense. "Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM)." Ver. 3.0. OO-ALC/TISE. May 2000. <www.stsc.hill.af.mil/gsam/guid.asp>.
12. Project Management Forum. <www.pmforum.org>.
13. Project Management Institute. <www.pmi.org>.
14. Project Management Knowledge Base. Extensive free library. <www.4pm.com>.
15. Defense Systems Management College. Project Manager Magazine. <www.dau.mil/forms/order_pm. asp>.
16. Software Program Managers Network. <www.spmn.com>.
17. Software Program Managers Network. Risk Radar software download. <www.spmn.com/rsktrkr.html>.
18. Software Program Managers Network Guidebooks. <www.spmn.com/products_guidebooks.html>.
19. Software Technology Support Center. <www.stsc.hill.af.mil.>.
20. TechRepublic Information Technology Forum. <www.techrepublic.com>.
21. Ten Step Project Management Process Site. <www.tenstep.com>.
22. Clinger-Cohen Act of 1996. The National Defense Authorization Act for Fiscal Year 1996.
23. Department of Defense. "Mandatory Procedures for Major Defense Acquisition Programs and Major Automated Information System Acquisition Programs." Regulation 5000.2-R. 10 June 2001. Part 2: 2.8 Support Strategy. Part 2: 2.9 Business Strategy. Part 5: Program Design.
24. Department of Defense. "The Defense Acquisition System, 4.5 Effective Management." DoD 5000.1.
25. Department of Defense. "Operation of the Defense Acquisition System, 4.7 The Defense Acquisition Management Framework." DoDI 5000.2.

About the Authors

Tim Perkins has been involved in software process improvement for the past 11 years, including leading the effort to initiate software process improvement at the then five Air Force Air Logistics Centers. As the software engineering process group leader at the Software Engineering Division at Hill Air Force Base, UT, he led the division in reaching Capability Maturity Model® (CMM®) Level 3. The division has gone on to achieve CMM Level 5. Perkins is Acquisition Professional Development Program Level 3 certified in Project Management and System Planning, Research, Development, and Engineering.

Software Technology Support Center
OO-ALC/MASE
7278 Fourth St. Bldg. 100
Hill AFB, UT 84056-5205
Phone: (801) 775-5736
Fax: (801) 777-8069
E-mail: tim.perkins@hill.af.mil

Roald E. Peterson is a senior systems engineer with Science Applications International Corporation. He has 22 years of electronic systems development experience, specializing in communications, architecture, and software development. Peterson was an editor and contributor for the "Guidelines for the Successful Acquisition and Management (GSAM) of Software Intensive Systems" and is the author of the "Condensed GSAM Handbook." He has a bachelor's degree in physics and master's degrees in computer resources management and electrical engineering.

Software Technology Support Center
Science Applications Int'l Corporation
920 W. Heritage Park Blvd. Suite 210
Layton, UT 84041
Phone: (801) 774-4705
Fax: (801) 728-0300
E-mail: roald.e.peterson@saic.com

Larry Smith is a senior software engineer and project manager for the Air Force's Software Technology Support Center at Hill Air Force Base. He provides software engineering, software process improvement, and project management consulting for the U. S. Air Force and other Department of Defense organizations as well as commercial and nonprofit organizations. Smith is a faculty member at the University of Phoenix. He is also certified by the Project Management Institute as a Project Management Professional. Smith has a bachelor's degree in electrical engineering and a master's in computer science.

Software Technology Support Center
OO-ALC/MASE
7278 Fourth St. Bldg. 100
Hill AFB, UT 84056-5205
Phone: (801) 777-9712
Fax: (801) 777-8069
E-mail: larry.smith4@hill.af.mil

Delivering Quality Products That Meet Customer Expectations

Louis S. Wheatcraft
Compliance Automation, Inc.

Why is it so difficult for project personnel to deliver a quality product on time and on budget that meets or exceeds their customer's expectations? A major contributor to project failure is neglecting to spend time at the beginning of the project on the basics. There are critical activities that must be accomplished and agreed to before writing requirements and beginning product development. These activities include clearly defining the project and product scope, including need, goals, objectives, drivers and constraints, assumptions, operational concepts, external interfaces, and feasibility and risk assessments. Unfortunately, many of these activities are often skipped. Developers jump into design without really understanding the reason for developing the product, and what it is they are supposed to do. This article focuses on one of the biggest problems in clearly defining your product scope: a failure to define and agree to the product's need, goals, and objectives.

Past articles in CrossTalk have cited dismal results for studies of software project failures and successes. In the April 2002 issue, Theron Leishman and Dr. David A. Cook reported the following facts regarding the Department of Defense software development:

> At the 5th Annual Joint Aerospace Weapons Systems Support, Sensors, and Simulation Symposium in 1999, the results of a study of 1995 Department of Defense (DoD) software spending were presented. Of $35.7 billion spent by the DoD for software, only 2 percent of the software could be used as delivered. The vast majority, 75 percent, of the software was either never used or was cancelled prior to delivery. The remaining 23 percent of the software was used following modification. [1]

Similarly in a July 1998 CrossTalk article, Lorin J. May reported on a Standish Group study of unsuccessful government software:

> According to the Standish Group, in 1995, U.S. government and businesses spent approximately $81 billion on cancelled software projects, and another $59 billion for budget overruns. Their survey claimed that in the United States, only about one-sixth of all projects were completed on time and within budget, nearly one third of all projects were cancelled outright, and well over half were considered "challenged." Of the challenged or cancelled projects, the average project was 189 percent over budget, 222 percent behind schedule, and contained only 61 percent of the originally specified features. [2]

One problem that contributes to cancelled projects is a failure to establish a shared vision of the final product at the beginning of the project. Having a shared vision requires defining the product's need, goals, and objectives before writing requirements and developing code. The purpose of new software or of an upgrade must be clearly understood before writing requirements, or divergent requirements will be written and important requirements will be missed. This vision must be provided to programmers before development to ensure that they maintain a single viewpoint.

"An error made by some project teams is the failure to differentiate the product scope from the project scope. Each defines overlapping but different need, goals, objectives, stakeholders, drivers, and interfaces."

Creating a shared vision is a basic concept in product development that is often neglected. There is an old saying that expresses this: "Failure to plan, is a plan to fail."

Project vs. Product

For the purposes of this article, I define a *project* or *program* as a system that consists of the people, processes, and tools that make up the environment in which a product or products will be developed. The product could be a hardware system, a hardware/software system, a process, or a service. The product is a separate system within the project or program system. Each may have its own scope.

An error made by some project teams is the failure to differentiate the product scope from the project scope. Each defines overlapping but different needs, goals, objectives, stakeholders, drivers, and interfaces. While the product will be driven by the project scope, there will be other drivers as well. The perspective is different when focusing on developing the product vs. managing the project or program. For large programs or projects, there typically will be a project manager as well as a product manager. The product manager may be referred to as the system engineer, the lead engineer, the lead software engineer, etc. His/her focus is on product management vs. project management.

This article deals primarily with the product. The following sections discuss the problem in clearly defining the product scope identified earlier – a failure to define and agree to the product need, goals, and objectives.

Clearly Define Product Scope

The scope of a product constitutes the vision: the need to develop or procure a product or service; the goals and objectives of the customer and your company; information about the customers and users; and how the product will be developed or purchased, tested, deployed and used [3, 4, 5]. The scope must unequivocally define the product boundaries. A product with no boundaries will diverge, or as Yogi Berra said: "If you don't know where you're going, you'll probably end up someplace else."

For example, you are assigned to lead a project team to upgrade a product. The main reasons for the upgrade are to fix known problems, to address lessons

Product (Implementation)	Need
New remote-piloted aircraft.	To provide low risk, accurate, real-time situational awareness of battlefield operations.
New software standard message protocol.	To enable a fighting force linked together as an integrated collection of interoperable systems.
New integrated intelligence software system that will collect and integrate terrorist information from multiple sources.	To quickly and accurately assess potential terrorist threats.
Integrated acquisition system.	To reduce the integration and maintenance overhead associated with current non-integrated, distributed systems.
A proactive tool provided to officers that ensures the accuracy of their promotion data prior to its submission to the primary promotion board.	To decrease the cost and use of resources in support of military promotion boards.

Table 1: *Examples of Defining a Product vs. Defining the Real Product Need*

learned, and to improve its operability, reliability, security, safety, and maintainability. The functionality of the current product is adequate to meet the basic need for the product; however, there are stakeholders that would like to add to current functionality and improve product performance.

While some stakeholders want to add a few *bells and whistles*, others have legitimate reasons to add to the product's functionality. These reasons include upgrading to new technology, making the product more competitive, making the stakeholder's jobs easier or more effective, and adding features based on changes to the product's operating environment. If you skip the up-front activities and fail to define and get an agreement on the scope of the product upgrade, there will be no clear boundaries of what is included in the upgrade and what is not. You are doomed to failure.

Getting agreement from all key stakeholders of the product scope before your team writes requirements and begins design ensures that everyone will clearly understand the requirements' boundaries for the upgrade. Involving key stakeholders in scope definition will avoid battles that result from differing visions and different interpretations of what should be included or excluded in the product. Issues can be identified and resolved before investing scarce resources into the requirements writing effort. Spending time to resolve issues, get questions answered, reduce debates, and confirm assumptions will result in reducing the time to write requirements, as well as speeding up the requirements review and baseline process.

Scope definition keeps requirements writers from diverging, reduces requirements inconsistencies, and keeps the big picture in view. An agreed-to product scope contributes to better requirements whose impact on development and testing is to avoid incorrect design and to reduce requirements discrepancies found in testing. Having a clear product scope will allow ground rules to be established.

In the preceding example, your first priority is clearly to maintain current functionality while specifically addressing the known problems and lessons learned as well as improving the current product's operability, reliability, security, safety, and maintainability. Requirements to add additional functionality or features will only be considered if specifically agreed to and documented; if they do not impact the development schedule and budget; and if they do not conflict with your efforts to improve the product's operability, reliability, security, safety, and maintainability.

Once product scope has been defined and agreed to by all key stakeholders, it must be formally baselined and controlled. Managing change to avoid *scope creep* is one of the biggest problems in government and industry. Too often, the scope of the software development project changes in midstream as stakeholders think of new features to add. One reason given for project cost or schedule overruns is a change in the agreed-to product scope without a corresponding change in cost or schedule. Change is inevitable. However, if new features are requested then the scope, which includes the cost and schedule, must be adjusted accordingly. Change is not free. Key stakeholders are often ignored until far too late in the process. By involving key stakeholders in product scope development and baseline, you will be able to minimize change. The ultimate payoff is reduced rework, reduced cost overruns, and reduced schedule slips.

Defining Need, Goals, and Objectives

The foundation of a product scope is having a clear understanding of the product's need, goals, and objectives. You do not want anyone on your team saying: "Why are we doing this?" You do not want everyone on your team having a different vision of what they are to develop.

In defining product scope, one problem you will frequently face is understanding product need, goals, and objectives and their relationships.

Product Need

It is the *need* that typically initiates a project. The need forms the basis of the project: The product is being developed to fulfill the identified need. The product need may come because of a new threat or opportunity, a mission or business need, a customer request, a technological advance, a deficiency in an existing product, or a legal requirement. The need may require a new product or an upgrade to an existing product. The product need is the driving purpose for designing and building or upgrading the product.

The need is why you are doing the work. For example, say I am going to build a house. Why? I need to protect my family. I am protecting my family from a number of things: the natural environment, persons or critters that might want to harm us, or persons who may want to steal our possessions. The basic need is to protect my family.

A common error when identifying the project need is to begin by stating an implementation (naming a product) rather than stating the underlying need for the project. The product is not the need; assumptions about need may be conflicting or just plain wrong. There are often several ways a need can be met. As an example, consider this statement: "We need a drill bit." What we *really* need is a hole. It may be that a drill bit is the best solution to obtaining the hole, but it is the hole that is needed.

One prime method to uncover the real need is to ask "Why?" If someone says they need a drill bit, ask "Why?" In this case, a drill bit is *not* needed, the hole is. If someone says they need a remote-piloted aircraft, ask "Why?" The real reason may be that they need to provide low-risk, accurate, real-time situational awareness of battlefield operations. In both cases, the drill bit and the remote-piloted aircraft are one of several solutions that will meet the need. A product development effort should be *need-oriented* and should *not* seek to justify a specific solution or acquisition program (see Table 1).

Identifying the problem the product is to solve is one of the best approaches to determining the real need. This point is clearly made by Robert Frosch, senior research fellow, Kennedy School of Government, Harvard University and

former NASA administrator:

> In my work here, my most frequent refrain is: "What problem is it you're trying to solve? What's the underlying question?" In my work on the Reports Review Committee of the National Academies, I have found that the most common problems faced by an advisory report at the end are generated by lack of clarity in setting the scope of the question at the beginning. [6]

For your project, you should be able to clearly define the specific problem the product is to address and demonstrate your understanding of the problem. Knowing and understanding the problem will enable you to explain why the project is worth doing, why the product is needed, and why the product is important to the customer.

In solving the problem, you can ask any or all of these questions: What is it you cannot do with the current system? What are the limitations that will prevent you from meeting the projected mission need? What is the key capability (benefits) that having this product will provide for your organization, the customer, or the product users? What will you be able to do differently if you have this product (in terms of improved productivity, operational effectiveness, or efficiency)? What will you not be able to do if you do not have this product? What are the consequences of not having this product (impact on customer/sponsor and/or users to perform mission responsibilities if the capability shortfall is not resolved, or impact of lost technological opportunity in terms of cost to your organization)? The answers to these questions can help you identify the real need for your product.

The process of answering these questions in the government arena for large projects is often accomplished during an activity called the *need assessment*. The outcome is a problem statement that summarizes the analyses and conclusions of the assessment. The resulting problem statement is referred to by various names: statement of need, need statement, or mission need statement [7 (a template for this process can be found here)]. From this problem statement, the need for the product can be identified, as shown in Table 2.

The need should be a short and concise statement. Once derived, the need should not change over time. If the need is changing, you do not know what is really needed, and you cannot build a product to meet a moving target. Do not let the real need be forgotten. It is the focus of your investment.

Once the need for the product is captured, identify its goals and objectives. When in a group, ask what the need is and you will often find a variety of needs stated. In some cases, this is due to a lack of agreement on what the need is. In other cases, they are listing goals for the product. If you find yourself creating a long list of needs, you are probably mixing goals and objectives with the true need. The challenge is to differentiate between the need and the goals and objectives.

Notice that I am advocating determining a *single* need. There might be times when a product will meet more than one need, but too many needs, like too many cooks, spoils the broth. The attempt to define a product to support multiple needs often results in a product no one can build, no one will buy, or no one can use. It does not mean your product does not have multiple features. Expanding that need into goals and objectives, developing operational concepts, and reviewing with stakeholders will result in many features that will become requirements to meet the need. Do not confuse features with the need.

Goals

In developing a product, ask what you hope to accomplish in meeting the need. *Goals* are the end toward which your efforts are directed. Each goal is tied to a part of the process in meeting the need. Goals are general responses to the need statement. Goals translate the need into a given solution to the problem. What will the project accomplish to affect the problem and meet the need?

There may be more than one way to meet a need. The goals you document will differentiate what you are going to accomplish vs. some other implementation.

Objectives

Objectives expand on the goals and state, in measurable terms, what you are trying to achieve. Objectives state the customer's expectations for performance. Objectives can include quality, new capabilities, needed functionality, etc. Objectives address these questions: In order to accomplish each goal, what specifically are you going to do? How will you know if you succeeded? What results do you expect? They may include cost and schedule objectives inherited from the parent project or program.

Both goals and objectives are short declarative sentences; goals are rather broad, and objectives fall under each goal

Table 2: *Examples of Problem Statement, Need, Goals, and Objectives*

Military Promotion Board Information Availability	**Military Logistics Command** [8]
Problem: There is no easy way for officers up for promotion to review the information contained in their promotion files. Because of this, there have been too many requests for supplemental promotion boards due to incorrect data in officers' promotion files used by the primary promotion board. This puts a burden on our personnel and promotion board workers.	Problem: Equipment is maintained at logistic centers that are distributed around the world. Command instructions specify how the data concerning the tasks and training of skilled mechanics and technicians doing this work will be managed. Currently, each logistic center has unique processes and software tools to do this. The costs of maintaining these individual software tools have become excessive. Command has dictated that the process and tools be standardized.
Need: Decrease the cost and use of resources in support of military promotion boards.	Need: Reduce the life-cycle costs of the command logistics process.
Goals: • Put the responsibility for checking officer promotion data in the hands of the officers being considered for promotion. • Provide officers a proactive tool to ensure the accuracy of their promotion data prior to the primary promotion board. • Eliminate excessive workload for the personnel office and promotion board personnel.	Goals: • Reduce the costs of maintaining multiple logistic systems with a standard, command-wide software system to be used at all logistic centers. • Provide an integrated training and tracking system for all logistics personnel.
Objectives: • Provide 24/7/365 access to members' official personnel records. • Provide correct display of data in officers' promotion files. • Remove personnel office workers as the middleman. • Provide up-to-date information to officers on corrective actions when errors are found. • Give officers correct and current promotion products. • Lower supplemental promotion board requests from 75 per board to 40. • Prevent fraud.	Objectives: • Use state-of-the-art computing, networking, and software technologies. • Deploy the system as a client-server system. • Allow supervisors to manage skills and training data in accordance with Command Instruction 21-108, Production Acceptance Certification. • Support the procedures to maintain Air Force Materiel Command Form 75, Job Knowledge-Training Certification Standard. • Interface to a command-wide training management system. • Develop a module to collect data for process improvement evaluation, also required by Command Instruction 21-108.

COMING EVENTS

February 10-13
Commercialization of Military and Space Electronics Conference
Los Angeles, CA
www.cti-us.com/ucmsemain.htm

February 24-27
Software Engineering Process Group Conference

Boston, MA
www.sei.cmu.edu/sepg/

February 25-26
Data Mining Technology for Military and Government Applications Forum
Washington, D.C.
www.worldrg.com

March 24-28
International Symposium on Integrated Network Management
Colorado Springs, CO
www.im2003.org

April 8-10
FOSE 2003
(Federal Office Systems Exposition)
Washington, D.C.
www.fose.com

April 28-May 1
Software Technology Conference 2003

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

May 3-10
International Conference on Software Engineering
Portland, OR
www.icse-conferences.org/2003

May 6-8
TechNet 2003
Washington, D.C.
www.technet2003.org

and are somewhat more specific.

It is common for people to confuse goals and objectives. Sometimes one or the other is defined without making a distinction. Sometimes the words are used together: *goals and objectives*. The important thing is that these topics and their content are addressed up front. One approach that can be used to differentiate goals and objectives is to think of a top-down approach. You start with defining the need. Then, you list goals. For each goal, ask the question "Why?" The answer should be the need. Then for each goal, list your objectives. Each objective should be traceable to one or more goals.

Once you have a draft of the product need, goals, and objectives, you must communicate them with the major stakeholders and get their agreement. The need, goals, and objectives form the foundation of your project's scope. Once you have a better understanding of the product, goals and objectives may change – but changes need to be closely managed. A change to goals or objectives is a change to the scope, which could impact cost, schedule, and risk.

Conclusion

To deliver a quality product, on time and on budget that meets your customer's expectations, get back to the basics and define, communicate, and get agreement on a clear vision for the product. To establish this vision, spend the time at the beginning of the project accomplishing and getting agreement on critical activities before writing requirements and beginning product development. These activities include clearly defining the project and product scope, including need, goals, objectives, drivers and constraints, assumptions, operational concepts, external interfaces, and feasibility and risk assessments. Going back to the basics of product management and establishment, getting buy-in, and communicating a clear vision for the project will lead toward a successful project. Anything less will compromise schedule, budget, quality, and mission success.◆

References

1. Leishman, Theron R., and David A. Cook. "Requirements Risks Can Drown Software Projects." CrossTalk Apr. 2002 <www.stsc.hill.af.mil/CrossTalk/2002/apr/leishman.asp>.
2. May, Lorin J. "Major Causes of Software Project Failures." CrossTalk July 1998 <www.stsc.hill.af.mil/crosstalk/1998/jul/causes.pdf>.
3. Hooks, Ivy F., and Kristin A. Farry. Customer-Centered Products: Creating Successful Products Through Smart Requirements Management. New York: Amacom 11 Sept. 2001.
4. Wheatcraft, Louis S., and Ivy F. Hooks. "Scope Magic." Nov. 2001 <www.complianceautomation.com/papers/scope_magic.pdf>.
5. Wheatcraft, Louis S. "The Importance of Scope Definition Prior to Developing Space System Requirements." INCOSE INSIGHT, Jan. 2002: 4:4 <www.complianceautomation.com/papers/scopedefincoseinsight.pdf>.
6. Robert Frosch. "Various." E-mail to Ivy F. Hooks. 6 Mar. 2002.
7. Federal Aviation Administration. "Acquisition System: FAA Mission Need Statement – Aids and Tools." <http://fast.faa.gov/archive/v0800/flowcharts/maflow/drfmnsaids.htm>.
8. Lynch, K. Edward. "AFMC Production Acceptance Certification and Depot Maintenance Quality Software Project." CrossTalk May 1998 <www.stsc.hill.af.mil/crosstalk/1998/may/afmc.asp>.

About the Author

Louis S. Wheatcraft has more than 34 years experience in the aerospace industry, including 22 years in the U.S. Air Force. During the last two years, Wheatcraft has worked for Compliance Automation where he conducts seminars on defining product scope, writing good requirements, and requirement reviews. He is a member of the Project Management Institute, INCOSE, and Toastmasters International. Wheatcraft has a bachelor's of science degree in electrical engineering from Oklahoma State University, a master's of arts degree in computer information systems, a master's of science degree in environmental management, and is completing a master's of science degree in studies of the future at the University of Houston, Clear Lake.

Compliance Automation, Inc.
1221 South Main St.
Suite 204
Boerne, TX 78006
Phone: (830) 249-0308
Fax: (830) 249-0309

Making Measurement Work

Cheryl Jones
U.S. Army

A successful measurement process becomes a way of doing business. Measurement is embedded in the organization, and performance improves because people are making fact-based decisions. This article describes characteristics of successful measurement programs using the Practical Software and Systems Measurement Initiative [1, 2] *guidance.*

Given the competitive importance of organizational performance and fact-based decision making, measurement programs have become increasingly important within the software and systems engineering communities. Measurement can no longer be implemented as a *check-the-box* process that is rolled out to satisfy a scheduled review or process improvement assessment. Today, measurement must provide real information to support critical project and organizational business and technical decisions, and the measurement results must be effectively communicated and used across the entire corporate entity.

The Navy provides us with two excellent examples of the impacts of using, and not using, measurement correctly. The F/A-18E/F program implemented a real-time management information system that communicated critical measurement-derived performance information to all project participants. This was instrumental in helping the F/A-18E/F program to be one of the most successful in recent memory. Conversely, on the Navy's A-12 program, objective information about project status was not available to the managers who were continuously required to deal with critical program constraints and performance shortfalls. Without accurate information, these managers were not able to properly make the difficult decisions required of them, and the program was ultimately cancelled.

Ten years ago, only a few organizations actually implemented measurement as an integral part of their technical and business processes. These forward-looking organizations established what worked well with respect to measurement, and even more importantly, what did not work. Under the umbrella of the Practical Software and Systems Measurement (PSM) Initiative, many software and systems measurement professionals that had successfully implemented measurement joined together in a government-industry-academia team to help put measurement into widespread practice. The goal of this PSM team was, and remains today, to help organizations meet a wide range of fact-based technical and business information needs by defining and helping them to implement a practical, information-driven measurement process.

Key Measurement Concepts

We all know how complex weapons, communications, and information systems are becoming. At the center of this growing complexity is the need to deal with continuous technical and business change. We need to make better and more timely decisions that will result in the success of our projects, systems, and organizations.

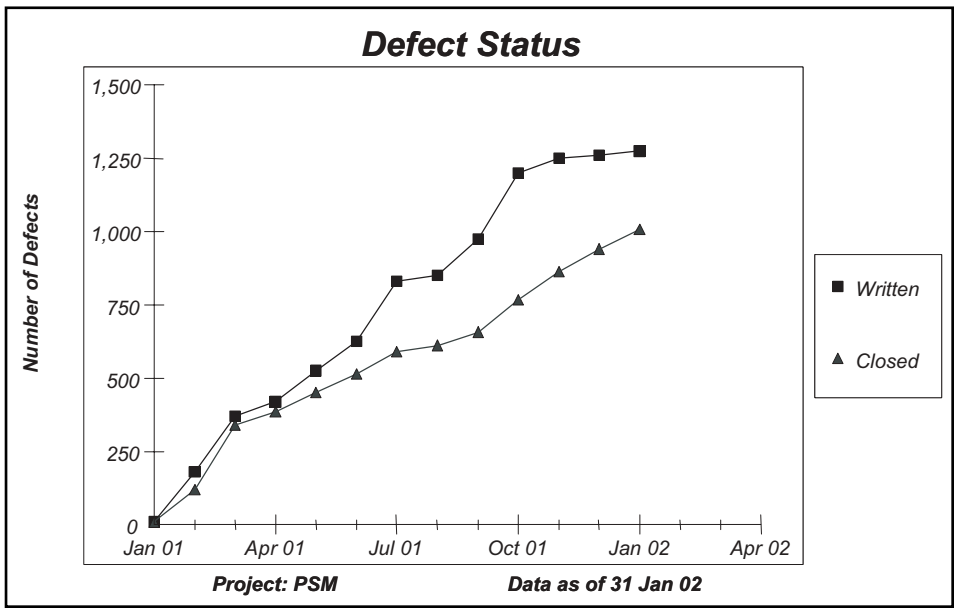
In today's environment, our measurement processes must provide objective information to support critical decision making in an ever-changing environment. As a simple example, the defect status indicator shown in Figure 1 was developed to help make decisions concerning test completion by evaluating product quality. In this example, the program manager needs to assess whether the system will be ready for a user acceptance test by April 2002.

As an entrance criterion for the user acceptance test, all open defects must be closed prior to the start of this test. An indicator was generated that included a graph of the number of defects written and closed, along with a calculation of the number of open defects remaining. The analysis indicated that the closure rate of defects has remained relatively constant, while the number of new defects appears to be slowing. This has led to a decreasing number of open defects, and provides a positive indicator that the product will be ready for user acceptance testing.

Obviously, this is a very simple indicator. More elaborate indicators would usually be generated that might include indicators with defect data by severity (generally all high priority defects must be closed, but some number of low priority defects may be allowed), by status (to allow an assessment of progress), by component (to identify defect-prone components that may require additional inspection or testing), or by age (to identify those that have been open a long time).

For those organizations that are just starting to measure their software and systems engineering processes and products, producing usable measurement results like this example can be challenging. Certainly the thought of starting up a measurement program has some considerable implications, especially when you need information immediately. The good news is that most successful measurement programs

Figure 1: *Quantitative Information Supports Critical Decision Making*



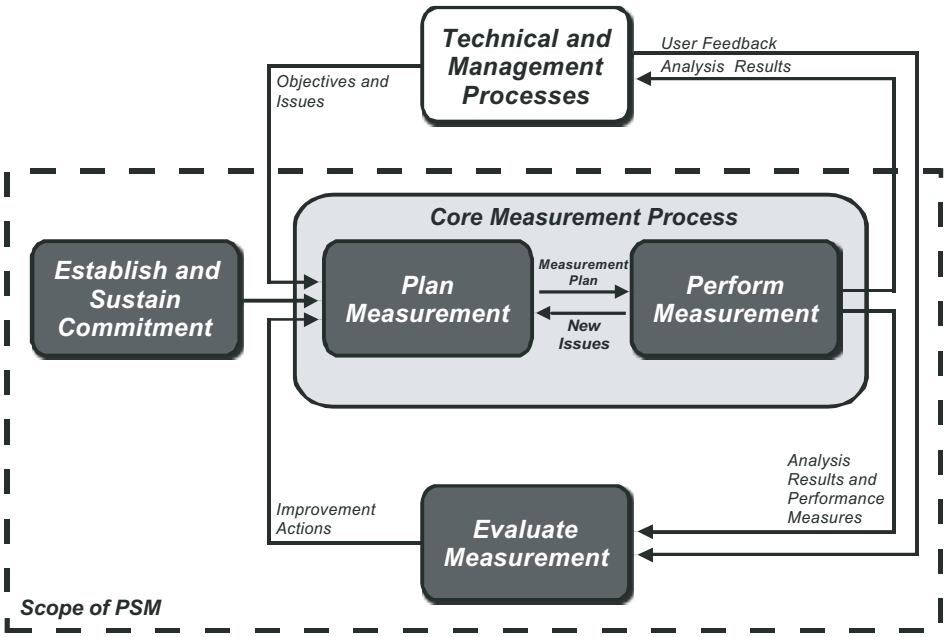


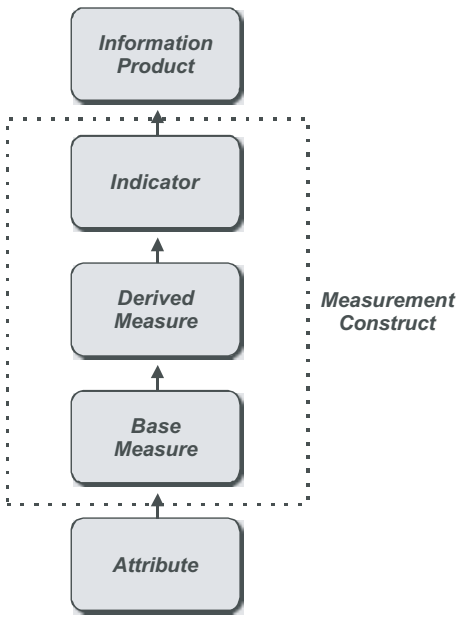
Figure 2: *Four Key Activities Are Characteristic of Successful Measurement Programs*

are based on a few manageable concepts. Together these basics provide the foundation for an effective measurement program even in the most complex environments, and they provide for a flexible, cost-effective approach to meeting defined information needs.

There are three recurring lessons learned from successful measurement programs. These are the basic building blocks of any successful measurement program:

- Measurement is a consistent but flexible process that must be tailored to the unique information needs and characteristics of a particular project or organization. Measurement needs to change as the environment changes around it. Changing information needs drive the measurement process.
- Decision makers must understand what is being measured. Key decision makers, including both the technical and business manager, must deliver value-added objective results that can be trusted on the day-to-day issues that these managers face.
- Measurement must be used to be effective. The measurement program must play a role in helping decision makers understand project and organization issues and to evaluate and make key trade offs to optimize overall performance.

Figure 3: *A Measurement Construct Relates What Is Being Measured to an Information Need*



Although these three basic measurement concepts seem like common sense, it is amazing how many organizations ignore them when they establish their measurement programs. Even with all of the change that they must deal with, some organizations still try to build their measurement programs around the *10 best measures* or try to measure so much that their measurement programs collapse under their own overhead burden. A large number of measurement programs fail early in their inception, usually because they do not provide information relevant to user information needs.

PSM has focused on delivering to both new and experienced measurement users guidance, tools, and other measurement products built around the foundational measurement concepts. A measure of PSM's success has been the adoption of its overall measurement approach by both the Capability Maturity Model® Integration℠ (CMMI℠) [3] and by the international software and systems engineering community, as embodied in the new commercial software engineering standard ISO/IEC15939 Software Engineering-Software Measurement Process [4]. An important aspect of the three initiatives is the consistent treatment of measurement by PSM, CMMI, and ISO/IEC 15939. The measurement practitioner now has an integrated guidance set based on real measurement experience that has proven successful in actual applications.

Since measurement success is so closely tied to the three basic measurement concepts, let us take a look at each of them more closely.

The Measurement Process

An underlying concept of measurement is that it should be flexible and tailorable based on the unique information needs and characteristics of each project or organization. Measurement must be iterative to support necessary changes that result from changing information needs and improvements in the measurement process itself.

The PSM process, shown in Figure 2, describes four activities that are part of a successful measurement program:

- **Plan measurement:** In this activity, measures are defined to provide insight into a project or organization's information needs. This includes identifying what the decision makers need to know, relating these information needs to those entities that can be measured, and then selecting and specifying prospective measures based on project and organizational processes. In the defect example in Figure 1, a comparison of the number of defects written and the number closed was used to address the question: "When will the system be ready for use acceptance test?"
- **Perform measurement:** This activity involves collecting measurement data, performing measurement analysis, and presenting the results so that the information can be used to make decisions. Analysis can include estimation, feasibility analysis of plans, and performance analysis of actual data against plans. For the defect example, the performance analysis included evaluating the trends of written and closed defects, and calculating test readiness.

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

- **Evaluate measurement:** In this activity, both the measurement process and the specific measures should be periodically evaluated and improved as necessary. For example, if the defect indicator does not provide enough information to adequately determine readiness for user acceptance testing, additional indicators may be added. The user may add an indicator of defect data by severity (generally all high priority defects must be closed, but some number of low priority defects may be allowed).
- **Establish and sustain commitment:** This activity involves establishing the resources, training, and tools to implement a measurement program effectively, and most importantly, ensuring that there is management commitment to use the information that is produced. In the defect example, if the measurement information is not used to develop plans for when user acceptance testing can begin, there is little need for collecting the data.

A measurement process that is flexible and tailored to project and organizational processes ensures that measurement is cost effective. Data should not be collected or reports distributed that are not needed or are not used. In addition, data collection and reporting should be automated whenever possible to provide an automatic by-product of normal project activity.

The process shown in Figure 2 provides a foundation for measurement for many disciplines, including software engineering, systems engineering, and process improvement measurement. An important thing to remember is that the same basic measurement process can support a wide variety of distinct and changing information needs in each of these areas. For additional information on the measurement process, see [3] and [4].

Connecting Information Needs to Actual Measures

A second basic concept in a successful measurement program is the communication of meaningful information to the decision makers. It is important that the people who use the measurement information understand what is being measured, and how it is to be interpreted.

PSM does this by incorporating a measurement information model that links the entities that are measured to the associated measures and ultimately to the identified information need. The measurement information model provides a structure for specifying how a particular information need will be addressed within the measurement process. This allows the measures to be clearly and consistently defined.

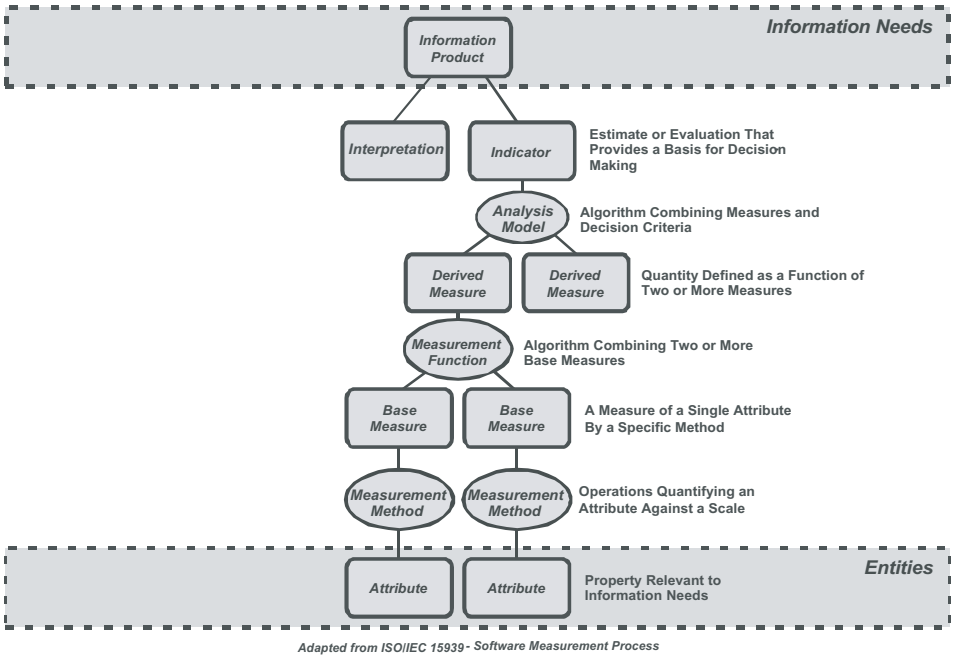


Figure 4: *A Measurement Specification Provides a Common Understanding of What Is Being Measured*

In the defect example, it is important for the decision makers to know exactly what the data represents in order to ensure that objective decisions are made. For the sample defect indicator presented, it is important for managers to understand that only identified defects are included, and that some number of latent defects will remain in the product. It is also important to ensure that planned and actual data are quantified using the same methodology so that the two sets of data are comparable.

The measurement information model provides a mechanism for linking information needs to what can be measured. A measurement structure, called a measurement construct, describes how the relevant attributes of products and processes are quantified and turned into indicators that provide a basis for decision making. The measurement construct may involve three levels of measures: base measures, derived measures, and indicators as illustrated in Figure 3. In our defect example, base measures include number of defects written and number of defects closed. The derived measure of open defects was then calculated as the difference between the two. The indicator presented is a graph of the two base measures.

For each of the base measures, derived measures, and indicators, additional information also needs to be specified about how the various measures are calculated. Figure 4 illustrates the structure of a complete measurement specification, including the measurement method, measurement function, analysis model, and decision criteria. These, along with the procedures for data collection, data analysis, and reporting provide an operational definition of a measure, addressing a specific information need. Figure 5 (see page 18) contains a high-level description of a complete measurement specification for the defect example described previously. For a description of each of these terms in more detail, see [3] and [4].

Defining the measurement terms to this level of detail provides everyone working with the data a common understanding of what is being measured, and how this relates to the information needs. This formalizes what is important. By detailing the base measures to be used, this also helps to highlight common measures that can be reused to address multiple information needs. This assists in prioritizing the measures to be implemented. Sample measurement specifications for many common measures can be found in the PSM guidance.

In the past, measurement terms were often defined in unique and inconsistent ways from organization to organization. This led to confusion and difficulty in widespread measurement implementation. In many cases, decision makers were unsure of what the measurement results actually represented. One of the advantages of the measurement information model is that it defines a consistent ter-

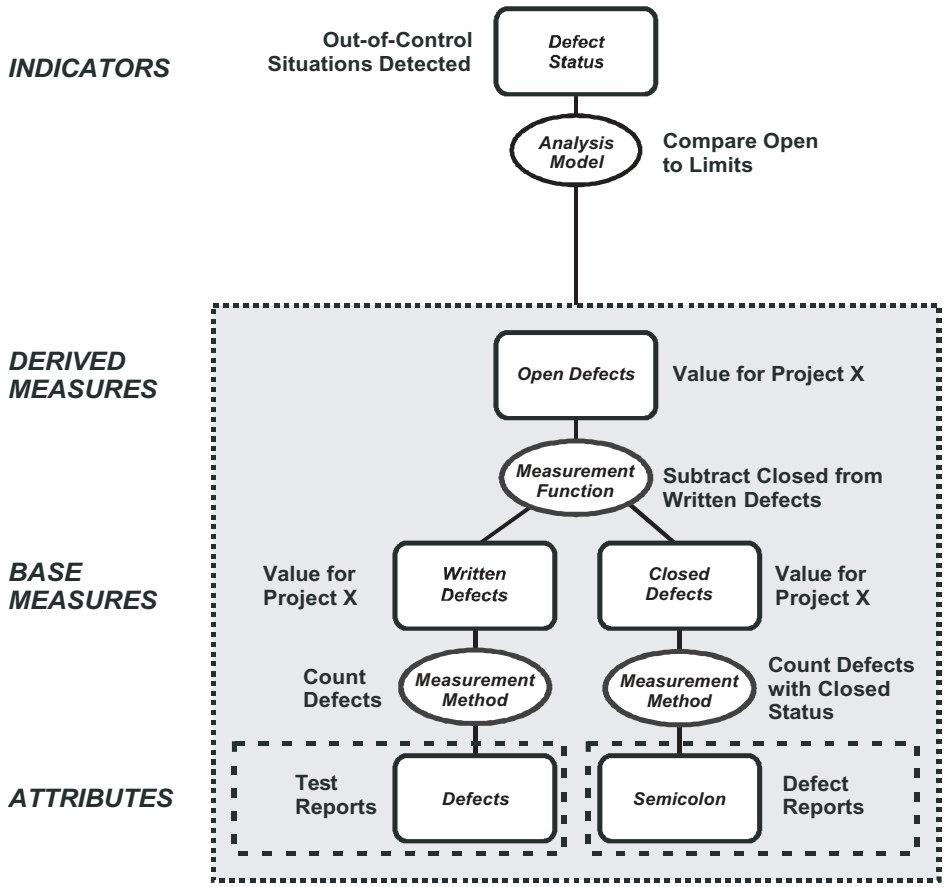


Figure 5: *A Measurement Specification for the Defect Example*

minology. This allows projects and organizations to document their information needs and selected measures to allow a common understanding of what is being measured. This is especially important as information needs change.

Understanding and Using Measurement Results

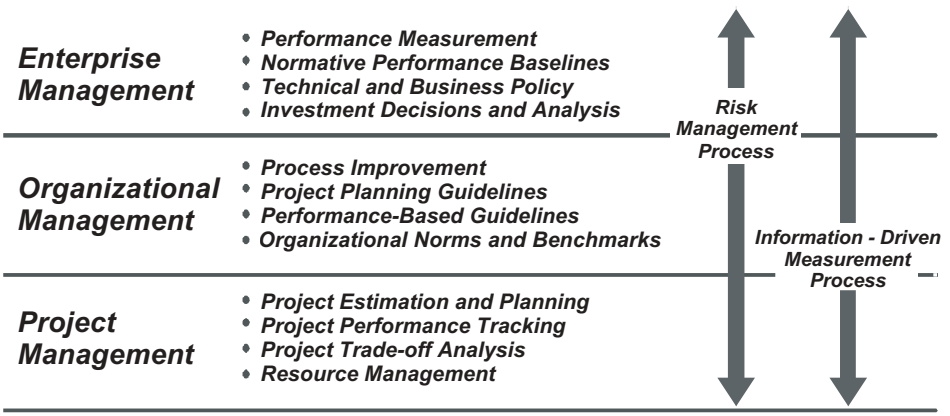
A third concept of a successful measurement program is that the measurement process is an integral part of the way business is conducted. In a successful organization, the measurement results are regularly used to make decisions. If the members of a project or organization are not able or willing to use measurement data to make decisions, the measurement program is of little use. In the defect example, if this measurement information is not used to develop plans for when user acceptance testing can begin, there is little need for collecting the information.

To support the use of measurement, information must be obtained early enough to allow managers to take the necessary actions to reduce risks or correct problems. Management decisions cannot wait for a complete set of perfect data to support management decisions, but should be derived from the minimum amount of data, complemented by real-time events and qualitative insight. Measurement should provide information on real-time events in a project, and facilitate communication.

The risk management and measurement processes should always be closely aligned. Risk management identifies the information needs that can impact project and organizational performance – information needs that should be objectively explained with the measurement results. The measurement data help to quantify risks, and subsequently provide information about whether risks have been successfully mitigated.

While measurement begins with a project-level focus, there are legitimate needs for information at higher levels of management. The information needs at different levels in an organization are related, as depicted in Figure 6. The project manager is concerned with the time and effort it takes to implement required product functionality and quality. At higher levels, managers are responsible for organizational performance and for improving organizational processes. At the enterprise level, managers are responsible for making investment decisions and ensuring performance is satisfactory. Since most organizations are composed of a portfolio of distinct projects, all of these uses of measurement rely on having good project-level information, aggregated to appropriate levels of the organization. The project level is where the processes and products are actually measured. As a result, a viable measurement program satisfies the needs of many levels of decision makers.

Figure 6: *All Levels of an Organization Have Measurement Information Needs*



Where to Get Help

The PSM project is a Department of Defense and U.S. Army sponsored initiative. The PSM project comprises a fully integrated approach of products and services. Products are developed and improved incrementally by a joint government, industry, and commercial technical working group based on actual implementation experiences. Products are updated based on the technical consensus of best practices and are freely provided. The project is supported by a number of transition organizations that are qualified to teach PSM concepts and transition the PSM measurement guidance. PSM's products include the following:

- "Practical Software Measurement" (Addison-Wesley).
- "Practical Software and Systems Measurement Guidebook" (PSM version 4.0).
- PSM Insight Tool (a PC-based measurement tool that allows tailoring to an individual project's needs).

PSM's Relationship to ISO Standards and CMMI

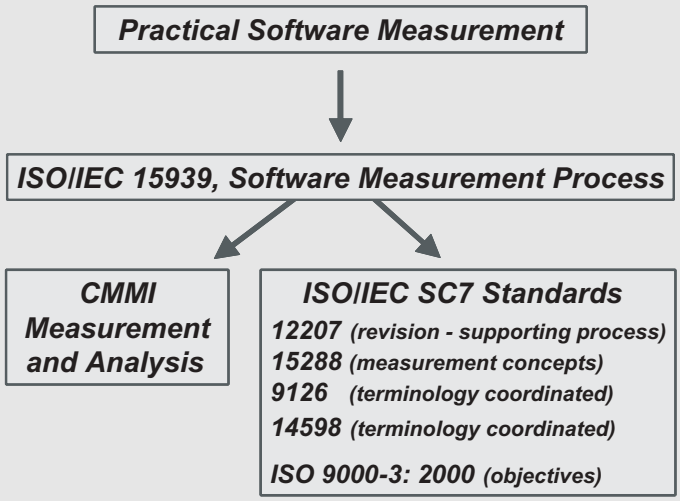
Practical Software Measurement (PSM), a product of a Department of Defense measurement initiative, served as the base document for the new international standard on measurement ISO/IEC 15939 Software Engineering-Software Measurement Process. The international standard describes the measurement process in terms of the purpose and outcomes of a compliant process, along with associated activities and tasks. The standard also defines the measurement information model and associated terminology.

PSM provides additional details on the activities and tasks presented in ISO/IEC 15939, and provides detailed steps to successfully meet these tasks. In addition, PSM provides detailed how-to guidance, including sample measures, lessons learned, case studies, and implementation guidance. PSM provides a set of sample measures using the measurement information model terminology. Both products are coordinated to provide users with a consistent framework for implementing a measurement program.

In addition, the purpose and outcomes of the measurement process from ISO/IEC 15939 have been added to the revision to ISO/IEC 12207 Software Life-Cycle Processes, within a new supporting process entitled Measurement. Measurement concepts have also been added to ISO/IEC 15288 System Life-Cycle Processes. The new measurement terminology has also been coordinated with the revisions to ISO/IEC 9126 Software Product Quality and ISO/IEC 14598 Evaluation of Software Products, so that all these standards will use a common set of measurement terminology once the revisions are complete. In addition, the purpose and outcomes of the measurement process have been added to ISO 9000-3: Application of ISO 9001:2000 to Software.

The draft international standard ISO/IEC 15939 in turn was used as an input to the measurement and analysis (MA) process area of the Capability Maturity Model® Integration℠ (CMMI℠) [4]. The MA process area provides a methodology for assessing whether a project's measurement program is compliant with the international standard, in addition to providing relevant information on CMMI-based process improvement activities. Overall, the CMMI helps organizations to institutionalize their measurement and analysis activities, rather than addressing measurement as a secondary function. In the MA process area, the activities of plan measurement and perform measurement are detailed in two specific goals that must be implemented and eight specific practices that are considered important in achieving the associated specific goals. The activities of evaluate measurement and establish and sustain commitment are considered through the generic goals, with elaborations specific to the MA process area.

The coordination of these documents means that the software and systems engineering communities have a consistent set of information-driven standards and guidance for implementing project and process measurement.



PSM, 15939, and CMMI Measurement and Analysis are Coordinated

- "PSM for Process Management and Improvement Guidebook" (PSM: MPM) .
- PSM technology papers (measurement with object-oriented development, spiral/evolutionary development, interoperability, and product-line architectures).
- Training and workshop materials.
- Supporting materials (descriptions of products and services).
- A qualified technical team to provide direct project support.

The guidebook and papers are available from the PSM Web site at no charge <www.psmsc.com>. We invite your participation in the PSM project, and your input into future work products. Please see the Web site for more information about how you can get involved.

Conclusion

A successful measurement process becomes a way of doing business. Measurement is embedded in the organization, and performance improves because people are making fact-based decisions. Three lessons learned from successful measurement programs include the following:

1. Measurement is a consistent but flexible process.
2. Decision makers must understand what is being measured.
3. Measurement must be used to be effective.

These measurement concepts described here are relatively easy to implement. There are many available resources to support their implementation.◆

References

1. Department of Defense. Practical Software and Systems Measurement Guidebook. v.4.0 Oct. 2000 <www.psmsc.com>.
2. McGarry, John, David Card, Cheryl Jones, Beth Layman, Elizabeth Clark, Joseph Dean, and Fred Hall. Practical Software Measurement –- Objective Information for Decision Makers. Addison-Wesley, 2002.
3. Software Engineering Institute. "Capability Maturity Model® Integrated℠ (CMMI℠) for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing Version 1.1 Staged Representation." Pittsburgh: Carnegie Mellon University, Mar. 2002.
4. ISO/IEC15939. Software Engineering – Software Measurement Process. 2002.

About the Author

Cheryl Jones is a software engineer for the U.S. Army Tactical Army Command, Armament, Research, Development and Engineering Center, responsible for measurement and analysis, risk management, and estimation. Jones is also the project manager of Practical Software and Systems Measurement and one of the authors of "Practical Software Measurement: Objective Information for Decision Makers."

U.S. Army TACOM-ARDEC
Bldg. 62
AMSTA-AR-QAT-S
Picatinny Arsenal, NJ 07806
Phone: (973) 724-2644
Fax: (973) 724-2382
E-mail: cljones@pica.army.mil

But I Only Changed One Line of Code!

Theron R. Leishman
Software Technology Support Center/TRW

Dr. David A. Cook
Software Technology Support Center/Shim Enterprises, Inc.

One of the basic concepts widely accepted as a software best practice is software configuration management. Although generally accepted, basic configuration management activities are often ignored, resulting in serious negative impact on software development and acquisition projects. This article is an introduction to basic, software configuration management concepts. It introduces these basic concepts and provides rationale for their implementation.

We are sitting in the Chicago O'Hare airport hours after the departure time of a flight home. After sitting on the plane for more than an hour, the pilot indicated that he was unable to get one of the engines to start. The pilot returned to the gate, and we were allowed to deplane. Eventually, it was determined that a hose in the engine was malfunctioning. This single hose was essential to the proper functioning of the engine. Fortunately, the engine mechanic was able to determine the hose that was causing the problem.

The mechanic understood the configuration of the plane engine and was able to refer to a defined list of all the engine parts, their relative arrangement to each other, and the methods to be used to assemble these parts into a jet engine. This is what is referred to as a configuration. The ability to properly distinguish the appropriate parts that must be used to build each type of engine is critical to assure consistent, safe production and use of aircraft engines.

As software developers, is it any less critical for us to be able to manage the assets that we create? The software we create is often maintained in various forms, with different tools, during its various stages of development, usage, maintenance, and operation. It is maintained at diverse locations, in diverse formats, and by diverse organizations. Software components are composed of parts, which are themselves software and are created using tools that are also software. Software systems are made of parts that cannot be touched, picked up, physically put in place, or manipulated. If you lose track of one of the software pieces, you have to re-create it. However, if you lose track of one of the tools, you *hope* that you can procure a new one. If the tool vendor is out of business – or no longer sells or supports the tool you need – you have a problem.

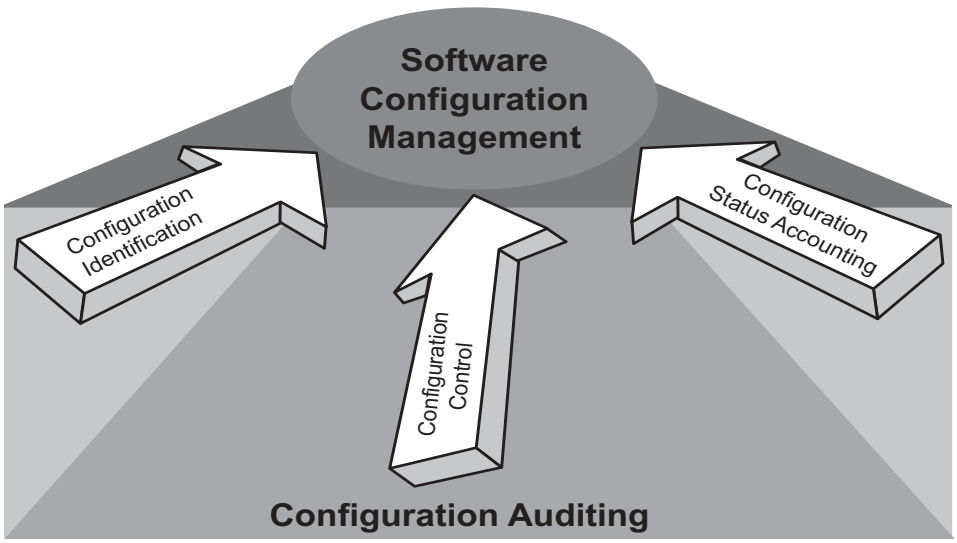
Software Configuration Management

Software Configuration Management (SCM) has been defined as the art of identifying, organizing, and controlling modifications to software [1]. SCM is a process that should be performed across the entire software development and maintenance life cycle. The configuration of software systems is by their nature complex. SCM may be described as a well-defined arrangement of software parts and the exact procedures and tools to be used for constructing the product or system from these parts. This must also include procedures for reconstructing various versions and releases [2]. The concepts of SCM are not new; numerous texts have been written on the topic of SCM. Typically SCM is defined by, and broken up into, the following four functional areas.

- Configuration Identification.
- Configuration Control.
- Configuration Status Accounting.
- Configuration Auditing.

SCM involves identifying what software assets are important to the organization. It includes controlling changes to these assets to ensure their integrity. Accounting for the status of these assets is important to the ongoing success of the organization. To assure that software assets are being properly accounted for, periodic configuration audits should be performed. Figure 1 depicts SCM's four basic functions. An introduction to each of these functions follows in this article.

Figure 1: *The Basic Functions of Software Configuration Management*



Configuration Identification

To properly manage all the elements of an aircraft engine, each of the critical parts must be properly identified. This is a key element in allowing proper management of the various engine configurations.

To provide the proper level of configuration management to software parts, we must also identify all the items that must be managed. These items, which we need to identify, are referred to as configuration items. Software configuration items (SCIs) are the items that are determined to be essential to manage the software product of concern. SCIs are the objects required to design, develop, build, maintain, test, and field a software product. SCIs are not things that can necessarily be shared between organizations, or even from one project to another. What an organization can, and should do, is develop documented criteria that can be consistently applied in determining which software-related items should be placed under configuration control.

Identification is the most important function of SCM. This is because items that are not identified cannot be managed. If you need *anything* to create your software product, then that item must be identified

and controlled. This includes compilers, operating systems, libraries, development tools, or environments. It might even include manuals and other documentation. It could also include such items as current spreadsheet programs, database programs, and word processing programs. In short, if you need it to accomplish your development task, then you probably should assign an identification number to it.

Configuration Control

As an inexperienced programmer, one of the authors of this article was approached by the primary stakeholder of an application for which he held maintenance responsibility. The stakeholder, a very influential individual in the company, wanted to have a small change made to a section of the application. Being pushed by the stakeholder to make the change, and being anxious to make brownie points in the company, the author made the simple, one line code change.

It was not until 2:00 a.m. the next morning, during the middle of a production run, that it was determined that the one line change had brought the entire application down. Not only was this application down, but also a critical corporate system that required input from the application was unable to complete essential processing while waiting for input.

Ouch! The desire to make a good impression in the organization backfired! The problem was that neither the organization nor any developers were controlling software assets. Anyone was allowed to make changes to the software at will. They were then able to have the changes migrated into a production environment with little or no unit testing, to say nothing of integration testing.

Software configuration control is the process of controlling and limiting changes made to software assets. According to the Institute of Electrical and Electronics Engineers, configuration control is an element of configuration management consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification [3]. In our words, configuration control is making sure that changes to software assets are only allowed to occur after they have been analyzed, evaluated, reviewed, and approved by a group authorized to control changes to software assets.

This group of people act as gatekeepers to control the flow of changes made to a SCI. They have authority to approve or veto proposed changes. This group is known as a change control board or configuration control board (CCB). The primary role of the CCB is to ensure that every requested change to an SCI is properly considered and coordinated prior to it being incorporated. The CCB should include representation from program management, systems engineering, software engineering, software quality assurance, software configuration management, independent test, and a customer representative.

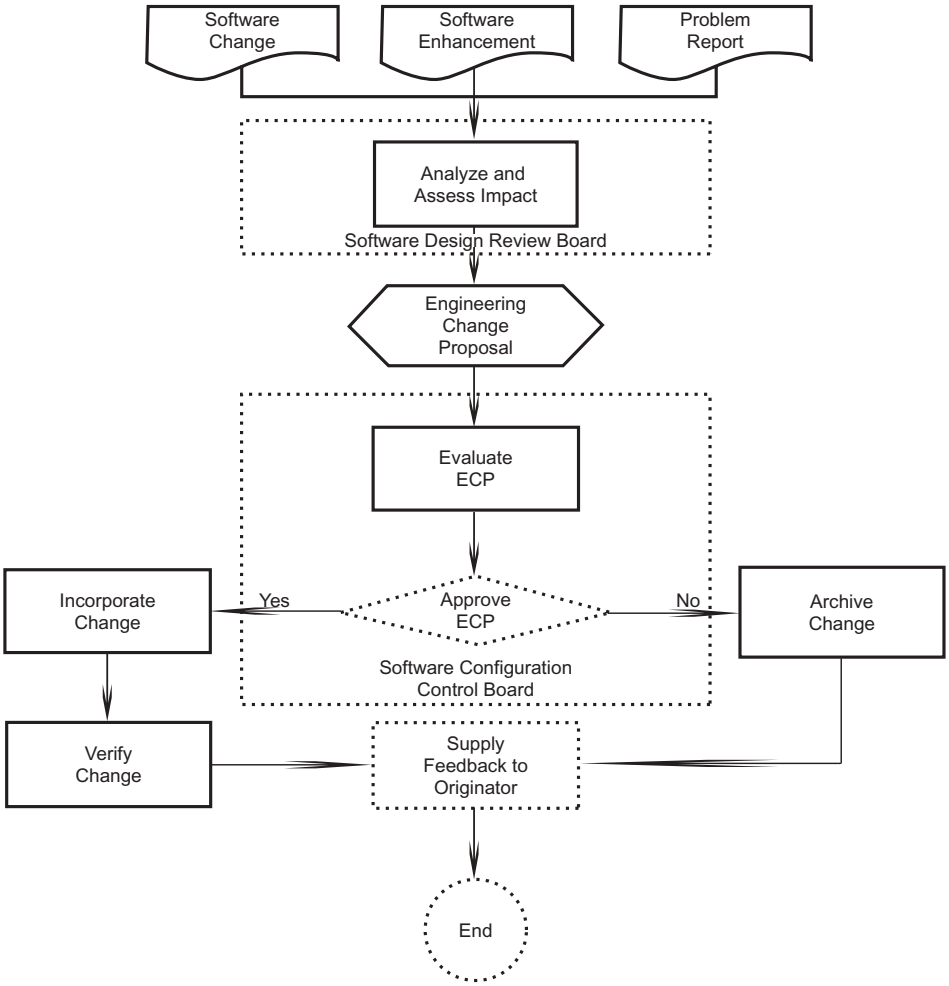


Figure 2: *The Software Change Process*

The CCB is responsible for seeing that all change requests are adequately reviewed, understood, and analyzed for impact prior to their decision to accept, defer to the next release, or decline the change request. Before any change to a SCI is made, the CCB ensures that someone did the research necessary to verify that the change will not unexpectedly impact other SCIs. The CCB, to make its decision, evaluates the impacts of each proposed change, and decides if the change is worth making.

Figure 2 presents a sample software change control process. A software design review board prior to the development of an engineering change proposal (ECP) reviews requested changes, enhancements, or problem reports. Once developed, the software CCB then evaluates ECP's. This board controls which ECP's are approved for incorporation into the software. Following approval and prioritization the change/enhancement will be made to the software.

Configuration Status Accounting

While attending college, one of the authors of this article was required to endure a course on cost accounting. In this course, he learned that cost accounting is that part of accounting that identifies, defines, measures, reports, and analyzes the various elements of direct and indirect costs associated with manufacturing and providing goods or services. Companies tend to care about things (assets) for which they have a financial investment.

Cost accountants are expected to track these corporate assets. They must know what the assets are, what costs were incurred in the development and ongoing maintenance of the asset, how the asset works with or interfaces with other assets, and be able to perform analysis on and report the status of these assets. To properly account for an organization's assets,

accountants must track the cost of each asset. In a manufacturing environment this may include assets that roll into other assets. The work, or product breakdown must be documented and all items tracked.

SCI's are the assets of a software product and must likewise be accounted for. Status accounting is the SCM activity responsible for tracking and maintaining data relative to each of these SCI's. It includes the tracking of changes to SCI's, and provides the ability to determine the status of each item at any phase of the software development or maintenance process. Status accounting involves gathering information to answer the following questions:

- What changes have been requested?
- What changes have been made?
- When did each change occur?
- What was the reason for each change?
- Who authorized each change?
- Who performed each change?
- What SCIs were affected by each change?

A repository of key software assets must be maintained to ensure that these key assets are properly accounted for. Changes to the status of these assets must be documented and tracked to allow the organization to know the status of any software asset at any point in time. Reports indicating the configuration of these software products must be defined and generated as a critical part of the accounting process. Following is a list of sample reports that the status accounting function should provide.

- Transaction log.
- Change log.
- SCI delta report.
- Resource usage report.
- SCI status report.
- Changes in progress report.
- Change request status report.
- Change completion report by developer, application, etc.

Configuration Auditing

There has been much interest lately in the integrity of certain large accounting firms who perform *independent* audits of corporate books. These audits are intended to attest to the soundness of the financial information reported by corporations.

In a previous lifetime, one of the authors of this article had the *opportunity* to work as an internal auditor. In this role he performed compliance audits, operational audits, proposal audits, and other types of audits in support of the mission of the organization. He was labeled by one acquaintance as Mr. Sneak & Peak. Auditors are not widely embraced as *good guys*. They are more likely considered those who go in after the war is over and stab the wounded.

By auditing adherence to policies, processes, and procedures, this corporation was able to demonstrate to their government customer that sound practices were consistently being followed in the delivery of the products and services for which the government was paying. In addition, people are more likely to follow the approved practices if they know someone is checking to see that they do. If developers gather metrics that nobody ever evaluates, the developers quickly learn that the metrics can safely be ignored. By the same token, if developers are never held accountable, then they quickly learn that policies, processes and procedures can be ignored. There must be accountability.

Software configuration audits are important for the same reasons. It is generally accepted that following good software development practices will contribute to consistent delivery of quality software products (assets). Audits should be conducted to help assure that software development policies, processes, and procedures are being consistently followed and adhered to.

In "Software Configuration Management for Project Leaders" presented at the 1997 Software Technology Conference in Salt Lake City, Tim Kasse explained that configuration audits verify that the software product is built according to the requirements, standards, or contractual agreement. Auditing also verifies that all software products have been produced, correctly identified, described, and that all requested changes are resolved.

A software configuration audit should be periodically performed to ensure that the SCM practices and procedures are rigorously followed. The integrity of the software baseline must be assessed. The completeness and correctness of the software baseline library contents must be verified. The accuracy of changes to the baselines must be verified to ensure that the changes were implemented as intended. It is recommended that a software configuration audit be performed before every major baseline change. Software configuration auditing should be continuous, with increased frequency and depth throughout the life cycle [4].

With regard to SCM, the terms functional configuration audit and physical configuration audit are often used. A functional configuration audit is intended to verify that the software functions as defined by the software requirements documentation. A physical configuration audit is intended to verify that all the items identified to be included in software release are actually included and that no additional items are included.

Who Cares?

OK, we have said lots of nice words about SCM, but why do you care about all this? Findings from various software assessments indicate that the lack of adherence to basic software development and acquisition sound practices continues to have a negative cost and schedule impact of software development, maintenance, and acquisition projects. SCM is one of these sound practices. Watts Humphrey said:

> The most frustrating software problems are often caused by poor configuration management. The problems are frustrating because they take time to fix, they often happen at the worst time, and they are totally unnecessary. [5]

If any of the following situations sound familiar to you, then you care!

- A software bug that was fixed six months ago has suddenly reappeared.
- A programmer just spent 12 hours making a change to the wrong version of the software.
- There is no way to trace requirements from the requirements document to the user documentation and source code.
- Two programmers working on a project have overwritten each other's code, rendering the last 40 hours of each of their work useless.
- No one can find the latest version of the source code.
- Fielded software that was working fine yesterday mysteriously will not work today.
- An application installed at various locations is running fine at some locations, but not at other locations.
- There is no way to evaluate impact on requirements from proposed changes.

Conclusion

In short, there is no such thing as a one-line change! Any software change is based on some type of requirement change. Proper identification of all the items critical to the development of software to satisfy requirements is essential to meeting those changing requirements. A supposed one-line change requires proper identification of what actually needs to change. It requires analysis of how the piece of code affected interfaces with other sections of the code. It requires analysis of what

impact a change to the code will have on other sections of the code. It requires review and approval by individuals who know and understand the application and can determine the extended impact of the requested change. It also requires documentation and traceability into what change is really required, who approved the change, when the change was made, and why it was made. It also assumes that someone is watching the process, that audits are being conducted assuring that the approved process for making changes is being followed, and that the software product matches the documentation.

Applying the concepts of SCM will improve the organization's ability to control software assets. Items and components essential to the development and maintenance of the software will be identified, managed, and controlled. Changes will not be made without proper analysis and approval. The status of software assets will be known and traceable at all times, and periodic audits will assure that the process is being followed. The probability of simple software changes hugely impacting projects will be greatly reduced.◆

References

1. Babich, Wayne A. Software Configuration Management. Mass.: Addison-Wesley, 1986.
2. Ben-Menachem, Mordechai. Software Configuration Management Guidebook. London: McGraw-Hill, 1994.
3. Institute of Electrical and Electronics Engineers. "Glossary of Software Engineering Terminology." IEEE Std-610.12-1990. New York: IEEE, 1993.
4. Kasse, Tim. Proceedings of The 9th Annual Software Technology Conference. Salt Lake City, UT. Software Technology Conference, 1997.
5. Humphrey, Watts. Managing the Software Process. Addison-Wesley, 1989.

About the Authors

Theron R. Leishman is a consultant currently on contract with the Software Technology Support Center at Hill Air Force Base, Utah. Leishman has 18 years experience in various aspects of software development. He has successfully managed software projects and performed consulting services for the Department of Defense, aerospace, manufacturing, health care, higher education, and other industries. This experience has provided a strong background in systems analysis, design, development, project management, and software process improvement. Leishman has a master's degree in business administration from the University of Phoenix. He is a Level 2 Certified International Configuration Manager by the International Society of Configuration Management, and is employed by TRW.

Software Technology Support Center
7278 4th Street
Bldg. 100
Hill AFB, UT 84056
Phone: (801) 775-5738
Fax: (801) 777-8069
E-mail: theron.leishman@hill.af.mil

David A. Cook, Ph.D., is the principal engineering consultant, Shim Enterprises, Inc. He is currently assigned as a software-engineering consultant to the Software Technology Support Center at Hill Air Force Base, Utah. Dr. Cook has more than 27 years of experience in software development and software management. He was formerly an associate professor of computer science at the U. S. Air Force Academy and also the deputy department head of the Software Professional Development Program at the Air Force Institute of Technology. Dr. Cook has published numerous articles on software process improvement, software engineering, object-oriented software development, programming languages, and requirements engineering. He has a doctorate degree in computer science from Texas A&M University, and he is an authorized Personal Software Process instructor.

Software Technology Support Center
7278 4th Street
Bldg. 100
Hill AFB, UT 84056
Phone: (801) 775-3055
Fax: (801) 777-8069
E-mail: david.cook@hill.af.mil

Risk Management Applied to the Reengineering of a Weapon System

Claude Y. Laporte
École de Technologie Supérieure

Guy Boucher
Oerlikon Contraves Inc.

In this article, a systems engineering process is briefly described followed by a discussion of the application of risk management practices to the reengineering of operator console stations of a missile weapon system. Lastly, 12 lessons learned are presented.

Oerlikon Contraves Inc. is a systems integrator specializing in the design, assembly, integration, testing, and delivery of complete systems solutions, including an air defense missile system. The system consists of a missile launcher mounted on a tracked vehicle or a fixed platform, together with radar and optical sensors, electronic control systems, and communication equipment.

The organization has been ISO 9001 certified since 1993. In 1997, it was also assessed at the Software Engineering Institute's (SEI) Capability Maturity Model® (CMM®) Level 2 [1] by independent assessors certified by the SEI. In addition to satisfying Level 2 goals, the organization also met eight of the 17 Level 3 goals.

In 1995, it was decided that a formal, systems engineering process had to be developed and implemented in order to seamlessly integrate disciplines associated with systems engineering. The development effort was initiated by performing an internal assessment of the organization's systems engineering practices. A decision was made to use as frameworks the CMM for Systems Engineering and the Generic Systems Engineering Process developed by the Software Productivity Consortium [2].

The systems engineering process (SEP) [3] describes management and technical activities, roles and responsibilities, and metrics and artifacts produced by each activity. The management activities of the SEP's major steps are summarized in Table 1 while Table 2 illustrates the technical activities (steps 210 through 270). The process had been applied to the reengineering of two subsystems: the launcher control electronics and the radar and electro-optical operator consoles [4].

"Dealing with formal risk management represented a mentality change not only for the project team but also for the entire organization."

Table 1: *The Management Activities of the Systems Engineering Process*

Major Steps	Substeps
110 Understand Context	111 Define Approach
	112 Estimate Situation
	113 Review Context
120 Analyze Risk	121 Perform Risk Analysis
	122 Review Risk Analysis
	123 Plan Risk Aversion
	124 Commit to Strategy
130 Plan Increment Development	131 Execute Risk Aversion
	132 Review Development Alternatives
	133 Plan Increment Development
	134 Commit to Plan
140 Track Increment Development	141 Monitor and Review Increment Development
	142 Update Increment Plan
	143 Review Technical Product
150 Perform Increment Closure	151 Baseline System Definition
	152 Assess Increment Closure
	153 Update External System Plan
	154 Commit to Proceed

The launcher control subsystem is composed of a main data processor that coordinates the operation of the sensors and the launch and guidance of the missiles, a missile tracker processor, a target tracker processor, and a servo control processor. The operator consoles consist of a radar and communication subsystems, and of an electro-optical console to control both the optical sensors and the missile launcher.

The Reengineering of Operator Consoles

The reengineering of the consoles was divided into two major increments: a system definition increment of the subsystem in its new configuration and a detailed hardware/software development increment, which was further broken down into several sub-increments. The identification of each increment was based on the nature of the deliverable products at the end of the increment. In both cases, the first increment deliverable would be a system requirement specification, and the second increment deliverables would be a set of design and equipment specifications plus a qualified working preproduction prototype.

The following paragraphs describe what was accomplished during increment one as well as what is being planned and performed for increment two. The emphasis of this article will be put on the risk activities that have been performed.

Overview of Increment One

Requirements Management

The system engineering CASE Tool CORE has been used to develop the console requirements. The database included the following types of information:

- Originating requirements (behavioral and nonbehavioral).
- Interface requirements.
- Verification requirements.
- Physical architectures.
- System diagrams.

The CORE database was baselined

after the completion of increment one.

Developing an Engineering Model

An engineering model was developed during increment one. The model ran on a standard PC, and its purpose was to show the new concept of operation and the proposed man-machine interface (MMI). The model was formally shown to stakeholders. Comments were collected and analyzed to modify and improve the system requirements in a second iteration.

Technology Search

A series of technologies related to either hardware or software has been researched and trade-off analyses have subsequently been documented. Many potential suppliers were met and a few employees attended real-time embedded conferences as well as virtual machine environment (VME) and high tech shows.

Training

Beside the training provided on the new systems engineering process, the only formal training provided had been on tools: the graphical user interface (MMI) CASE tool, and CORE, the system definition CASE tool. Training was also later performed on VxWorks operating system, Rhapsody software development CASE tool, and unified modeling language software development methodology.

Overview of Increment Two

The plan for increment two consisted of proceeding with both the hardware and software detail design based on the interim system definition and the engineering model generated during increment one. The detailed development will include the construction of an engineering unit to support the hardware and software development and the construction of a preproduction unit that will support system integration and qualification activities. In addition, simulators will be built in parallel to support development, integration, and validation efforts.

The plan for increment two also included other nonrecurring activities such as the production jigs, tooling and logistic activities, technical publications, and training.

The Application of Risk Management Activities

SEP Step 120: Analyze Risk

In SEP step 120, risks were analyzed, risk mitigation strategies were developed, and stakeholders' commitment was made on mitigation strategies (Substeps 121 and 122). The high-level process, as illustrated in Figure 1 and Table 3 (see page 26), describes what risk management activities should be performed, but it does not prescribe any particular method. The members of the project were aware of the method used by software engineers since a method was described in the project planning and tracking activities of the company software engineering process. After a brief discussion, the team decided to use the method proposed by the U.S. Air Force [5]. At the beginning of the project, it was felt that this step looked like a paper exercise and was not very useful. However, it was the first development project to proceed with a formal method to handle risks.

Major Steps	Substeps
210 Analyze Needs	211 Determine Stakeholders
	212 Define Problem Domain Assess Problem Needs and Constraints
	213 Define Environment
	214 Develop Informal Functionality
220 Define Requirements	221 Determine Behavioral Requirements
	222 Determine Performance Requirements
	223 Map Behavior to Performance
	224 Refine Requirements
230 Define Functional Architecture	231 Partition Requirements into Functions
	232 Define Lower Level Functions
	233 Define Functional Interfaces
240 Synthesize Allocated Architecture	241 Allocate Functions to Alternative Solutions
	242 Define Physical Parameters
	243 Define Physical Interfaces
	244 Integrate Design
	245 Refine Physical Architecture
250 Evaluate Alternatives	251 Assess System
	252 Perform Sensitivity Analysis
	253 Allocate Performance to Technical Parameters
	254 Assess Technical Risks and Problems
	255 Identify and Perform Trade-off
	256 Select Best System Solution
260 Verify and Validate Work Products	261 Define Verification and Validation Procedures
	262 Validate System
	263 Verify System
270 Release System Definition	271 Control Technical Decision Data
	272 Control System Configuration

Table 2: *The Technical Activities of the Systems Engineering Process*

A risk management plan (RMP) was developed containing two main sections. The first section described the program overview and defined terms based on the following:

- Type of risk (cost, program, schedule, technical, and supportability).
- Assessment of risk impact (catastrophic, critical, marginal, and negligible).
- Overall categorization of risk (high, moderate, and low).

The RMP specified who was responsible for the risk management and how the risks were to be managed during the increment. This section was quite generic; it could be reused by other projects.

The second section of the RMP was specific to the project. It was mainly composed of a single matrix that listed all of the identified risks. The risk identification process was performed through brainstorming sessions with both the development team members and stakeholders. Along with the list of risks in the same matrix was the following information:

- Type of risk (cost, program, schedule, or technical).
- Probability of occurrence (very low, low, medium, high, or very high).
- Impact (catastrophic, critical, marginal, negligible, and cost).
- Overall risk (high, medium, low, and cost).
- Identification of impact on other projects.
- Brief resolution plan.
- Drop-dead date.
- Person(s) responsible (member of the project team, functional manager, project manager, or engineering director).
- Hours or resources required performing the project.
- Resolution status (open or close).

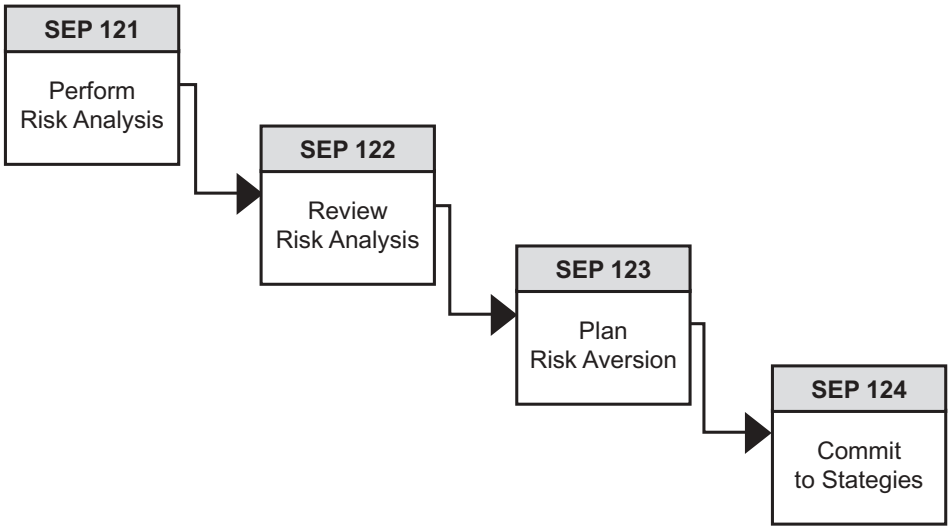


Figure 1: *Risk Management Activities*

The Implementation of the Risk Management Plan

The actions and status of the risks were then reviewed on a weekly basis during project reviews. When a mitigation plan was required, e.g., special resources and a considerable amount of hours, then specific risk activities were directly integrated in the detailed work breakdown structure and scheduled like any other major development items.

Some of the risks identified were as follows: project risks such as budget overrun, schedule delays mostly due to lack of dedicated resources, and technical risks such as the lack of experienced personnel in using a new process, and a new CASE tool (CORE). Also, since this project was performed concurrently with another project, it was necessary to closely monitor integration, validation and verification activities, and interfaces definition with the rest of the missile system. Finally, specific risks like availability of commercial off-the-shelf hardware, mastering of new technologies such as VME, development of a new custom circuit card assembly, and development of new communication bus were also identified.

Risk impacts were represented by a weighted probability of occurrence and consequence index. This risk matrix was stored in a database and was continuously updated during the two increments.

In some cases, the same mitigation strategy addressed several risks. Mitigation strategies included activities such as pilot projects, engineering models and mock-ups, additional analyses, and subsystem modeling. Specific participant training was also planned in some areas. Finally, a formal review with stakeholders helped to identify other risks, gather mitigation suggestions, and obtain final commitment (substep 124).

Lessons Learned

Quantification of Risks Issues

During increment one, risk only had a *qualitative* score, i.e., high, medium, or low. We found that this had two major drawbacks compared to quantitative evaluations:

- It did not have the same weight or necessary attention from management.
- No money/resources were set aside should the risk issue have occurred. This could lead to budget overruns.

Evaluation of Risks in a Systemic Perspective

For increment two, we quantified and costed all risks, even the ones that the team had no control over such as hiring or allocating budgets and expenses. The development team would ultimately be impacted should a risk occur. As a result, the company decided to put money aside for risks in the budget for increment two.

Risk Management Is Not Free, But It Is a Wise Investment

We quickly found out that some risks required a lot of effort to mitigate. One example was the activity related to the engineering model in increment one. It was decided to proceed with an engineering model to mitigate a risk previously identified that related to the fact that we had no customer requirement. The fear was then that we would proceed with a design that would not meet any potential customer wishes.

Approximately 800 hours were spent to model a new concept of operation and an MMI. This included activities such as model design and, even more important, validation of the concepts with a selected group of operators from inside and outside the company.

This model allowed us to develop and refine the system requirements as well as define software use cases with a very high confidence level that they would remain stable throughout the entire design chain. Although it was difficult to precisely assess the amount of time/money that has been and will ultimately be saved, one can imagine what would be the cost of delivering a product that would not meet customer expectations.

Another example is a pilot project performed in increment two. This pilot project came as a result of a risk identified that expressed the concerns that we would enter the software design phase with a new methodology, new CASE tools (design and GUI), and a new development environment. About 1,000 hours were spent on a mini-project that had the main objectives to verify the capabilities of the tools, to verify the integration of the tools, and to propose a design method.

The results and conclusions obtained through the development of this pilot project were crucial to generating a proper software development plan that needs

Table 3: *The Risk Activities of the Systems Engineering Process*

120 Analyze Risk	121 Perform Risk Analysis	Identify Potential Risks
		Identify Potential Loss and Consequences
		Analyze Risks Dependencies
		Identify Risks Probability of Occurrence
		Prioritize Risks
		Identify Risk Aversion Strategies for Each Risk
	122 Review Risk Analysis	Review Risk Analysis
		Identify Risks to Be Part of the Risk Management Plan (RMP)
	123 Plan Risk Aversion	Define a Risk Monitoring Approach
		Estimate Risk Aversion Strategy Cost and Schedule
		Recommend Risk Aversion Strategies
	124 Commit to Strategy	Obtain Stakeholders' Commitment

to clearly show, organize, and plan the work of a group of more than 20 persons for a 24-month time frame. The pilot project represented about 1.5 percent of the total software design effort, but it was sure worthwhile since it ensured that the remaining 98.5 percent of the project would be done properly and correctly.

Pilot Projects As a Risk Mitigation Strategy

It was very important to carefully select pilot projects and their participants since these projects would foster adoption of new practices throughout the organization. Also, first-time users of a new process would make mistakes; it was therefore mandatory to properly coach the participants. If participants sensed that mistakes would be used to learn and make improvements to the process instead of *pointing fingers*, the level of anxiety was reduced. This also led individuals to bring forward suggestions instead of *hiding* mistakes. Most of the participants for both projects were therefore selected within the working group who developed the SEP. Other participants were given a two-day training session on the SEP.

Management's Response to Risks

Dealing with formal risk management represented a mentality change not only for the project team but also for the entire organization. Yet, when risk management activities were done properly by the development team, management was more prone to agree and support the risk activities that resulted from the risk analysis.

Risk Mitigation Leads to Design Decisions, Development Strategies

The results of the risk mitigation activities related to technical risks will necessarily lead to, or as a minimum be an input to, design decisions and will provide direction for follow-on activities. In fact, whether a mitigation plan arose from generating an analysis, conducting a test, or constructing a physical or behavioral model, the result will be the confirmation of a hypothesis or the identification of the best design alternative. Ultimately, this leads to design decisions and subsequent development strategies.

Training as a Risk Management Issue

One important aspect of risk management was training. Previously, most plans showed a nice flow-down of activities with associated efforts, as it should be. However, these plans also reflected the fact that they were all conducted by highly skilled personnel that knew exactly what to do at all times. This obviously did not represent reality. Therefore, appropriate training became mandatory to manage the risks, and training activities were built into the project plan.

Dividing a Project Into Increments As a Risk Management Strategy

Project increments must be carefully defined so that they remain manageable. Their associated activities were not too long to be properly tracked, and on the other end were not too small so that their activities required micro-management. Project manager experience was found to be a critical asset for project and increment definition. A manageable increment size was also critical for the proper performance of design reviews; in those reviews, participants kept their focus on the increment scope.

"It was very important to carefully select pilot projects and their participants since these projects would foster adoption of new practices throughout the organization."

New Process Implementation Risks

It was found that for some areas of the SEP, specific deliverables were difficult to determine precisely. This situation happened because the end products (i.e., project documents) grew iteratively as process steps were performed. It was therefore difficult to closely measure the progress of the activities and report progress to management. As a result, *lessons learned* were generated, and this led to the development of a specific set of methods/instructions to support the project manager and his team by providing better definitions and tracking/reporting methodologies. The lessons learned and the various instructions have been distributed and electronic copies are available on the company intranet.

Risk Associated With People Issues

Managing the human dimension of the project was found to be an element that not only fostered the adoption of the new process, but also created an environment where changes were introduced at an increasingly greater rate. Members of the engineering organization realized that managing the *soft stuff* was as important as managing the *hard stuff*. Additional information about managing people issues can be found in a previous CrossTalk [6].

Risk Management Activities Are Planned and Included in the Project Plan

Since a substantial amount of energy was expended in *risk management* activities, those activities were identified, estimated, and incorporated in the project plan. It is important to note that risk management is part of the standard company work breakdown structure and a level of effort is estimated and planned accordingly. In addition, as the major risk mitigation strategies become part of the system plan to be approved by the organization, commitment is established. The costs associated to that risk effort and associated mitigation strategies are then tracked as any other work breakdown structure activity.

Appointing a Project Risk Officer

When a project is composed of many projects similar to the one described in this article, all risk activities may represent a substantial effort. Also, the risks have to be analyzed at the project level since risks in one subproject may create risks at the project level. Risks from different subprojects may be analyzed and mitigated at the project level instead of being mitigated individually. It was found that all project risk activities were better managed by one individual. A project role called risk officer had been established. The risk officer hat was allocated, as a secondary duty, to a member of the team who was interested by this role and had a lower load in the project.

Conclusion

A new SEP involving managing risks had been deployed and used in the re-design of a missile system operator console. The risk management activities were found to be very useful to plan activities and collect technical and managerial information more formally in the course of the projects. It also helped manage and improve the dynamic human dimension of the development project.◆

References

1. Paulk, M. et al. Capability Maturity Model for Software. Pittsburgh: Software Engineering Institute, 1993.
2. Software Productivity Consortium. A Tailorable Process for Systems Engi-

neering. Software Productivity Consortium, Jan. 1995.
3. Laporte, C. Y., and N. R. Papiccio. Development and Integration of Engineering Processes at Oerlikon Aerospace. Proc. of the Seventh International Symposium of the INCOSE. Los Angeles, CA, 1993.
4. Laporte, C. Y., A. Guay, and J. Tousignant. The Application of a Systems Engineering Process to the Reengineering of an Air Defense System. Proc. of the Eighth Annual International Symposium of the INCOSE. Vancouver, British Columbia, Canada, 26-30 July 1998.
5. U.S. Air Force. USAF's Software Risk Abatement Handbook. AFSC/AFLC Pamphlet 800-45, 30 Sept. 1988.
6. Laporte, C. Y., and S. Trudel. "Addressing the People Issues when Developing and Implementing Engineering Processes." CrossTalk Nov. 1999.

Additional Reading

1. Forsberg, K., and H. Mooz. Application of the 'Vee' to Incremental and Evolutionary Development. Proc. of the Symposium of the International Council on Systems Engineering. St. Louis, MO, July 1995.

About the Authors

Claude Y. Laporte is a software engineering professor at the École de Technologie Supérieure in Montreal. He was an officer in the Canadian Forces and retired at the rank of major. He joined Oerlikon Contraves Inc. in 1992, then called Oerlikon Aerospace, where he coordinated the development and implementation of engineering and management processes. Laporte has a bachelor's degree in science from the Canadian Military College of Saint-Jean, a master's of science degree in physics from the Université de Montreal, and a master's degree in applied sciences from the Department of Electrical and Computer Engineering at École Polytechnique de Montreal.

École de Technologie Supérieure
1100 Notre-Dame Ouest
Montreal, Quebec
Canada, H3C 1K3
E-mail: claporte@ele.etsmtl.ca

Guy Boucher is a project manager on various projects at Oerlikon Contraves Inc. He was formerly radar principal engineer at the company. Boucher served the Canadian Forces for a period of five years as a radar engineer on a long-range radar station and as a Canadian representative on a joint development program of the U.S. Over-the-Horizon-Backscatter Radar in Bangor, Maine. He left the Forces in 1987 at the rank of captain. Boucher has a bachelor's degree in electrical engineering from the Royal Military College of Canada.

Oerlikon Contraves Inc.
225, boul. du Séminaire Sud
Saint-Jean-sur-Richelieu, Quebec
Canada, J3B 8E9
E-mail: gboucher@oerlikon.ca

WEB SITES

Software Technology Support Center

www.stsc.hill.af.mil

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

Risk Management

www.acq.osd.mil/io/se/risk_ management/ index.htm

This is the Department of Defense (DoD) risk management Web site, a working group composed of representatives from the services and other DoD agencies involved in systems acquisition to assist in the evaluation of the department's approach to risk management. The working group will continue to provide a forum for sharing experiences and knowledge in order to provide program managers with the latest tools and advice on managing risk.

Software Program Managers Network

www.spmn.com

The Software Program Managers Network (SPMN) is sponsored by the deputy under secretary of defense for Science and Technology, Software Intensive Systems Directorate. It seeks out proven industry and government software best practices and conveys them to managers of large-scale DoD software-intensive acquisition programs. SPMN provides consulting, on-site program assessments, project risk assessments, software tools, guidebooks, and specialized hands-on training.

Software Engineering Institute

www.sei.cmu.edu

The Software Engineering Institute (SEI) features information on "Building High Performance Teams Using Team Software Process℠ (TSP℠) and Personal Software Process℠ (PSP℠)." The SEI is a federally funded research and development center sponsored by the U.S. Department of Defense The SEI's core purpose is to help others make measured improvements in their software engineering capabilities.

The Software Productivity Consortium

www.software.org

The Software Productivity Consortium is a nonprofit partnership of industry, government, and academia. It develops processes, methods, tools, and supporting services to help members and affiliates build high-quality, component-based systems, and continuously advance their systems and software engineering maturity pursuant to the guidelines of all of the major process and quality frameworks. Based on the members' collective needs, its Technical Program builds on current best practices and information technologies to create project-ready processes, methods, training, tools, and supporting services for systems and software development.

High Quality, Low Cost Software Inspections

Louis A. Poulin
GRafP Technologies

What do inspections, peer reviews, walk-throughs, and structured reviews have in common? These are all terms that are used interchangeably in software engineering. Yet, the activities that they entail are rarely carried out consistently in the course of developing an application. This article reviews this theme as Ronald A. Radice presents it in his new book.

I would bet that all articles submitted to CrossTalk, including this one, for which I can vouch, have been peer-reviewed. Why is it that peer reviews in the publishing industry are widely accepted, while they are the first items to be dropped off the priority list in the software industry?

According to Ronald A. Radice in his book "High Quality Low Cost Software Inspections" [1], there are several reasons why inspections are not more widely used in software development. First is the belief that inspections can only be done one way, a myth this book has all but obliterated.

Second is that inspections are not easy to do well, given the psychology that permeates them. Radice addresses this topic by discussing participant personalities such as aggressive inspectors, intimidating moderators, weak moderators, and defensive producers – whose products are being reviewed – and offers suggestions on how to deal with these situations.

Third, the perception that inspections represent an added cost to software development is still widely prevalent. The book certainly helps in countering this argument, with plenty of charts and data that demonstrate the added value of inspections. However, Radice, may very well have identified the fundamental underlying cause software inspections are not more widely used: inspections are low tech and are not the most enjoyable engineering tasks, especially when compared to design and coding.

But software inspections do work, and Radice's book contains 400 pages that not only demonstrate their value but also offer various approaches, techniques, and guidelines to conduct them. "High Quality Low Cost Software Inspections" is a must for anyone wishing to start inspections in their organization or to those who have performed inspections for some time and want to get better results. Radice describes the inspection process in detail, including the roles assumed by inspection participants and the type of data that should be collected, all the way to causal analysis of defects detected through such reviews.

Inspections also contribute to the culture change experienced by software companies that appreciate the value of data and allow the data to be used safely, in a nonthreatening way by the people who provide the data. However, this is easier said than done and does not happen overnight. The book includes a chapter on managing inspections and another on practical issues you can expect to deal with when introducing inspections. These chapters will prove helpful in preventing lukewarm reception by those who have been identified as participants, or downright failures.

> *"... inspections are low tech and are not the most enjoyable engineering tasks ..."*

The chapter on economics of inspections is particularly eloquent for anyone who needs to be convinced of their value. It references Infosys, where two teams were set up to assess inspections and unit testing. Inspections found 2.7 times more defects than did unit testing. According to Radice, another feature that differentiates inspections from unit testing is that when defects are found in inspections, the fix is often understood as soon as the defect is identified. Testing is characterized by a more serial approach: After a defect symptom has been observed, its cause must then be sought out and a fix devised.

Radice also takes a jab at the Software Engineering Institute's Capability Maturity Model® (CMM®) Integration℠ (CMMI℠) for diluting the value of inspections. Whereas peer reviews were deemed important enough to deserve a whole process area in the CMM for Software, they have now been reduced to a goal within the Verification Process Area in the CMMI. Implementation of inspections with the CMMI is now more a matter of choice than a requirement. Potentially, organizations that do not see a need to perform inspections will now have a bigger hole to squeak through to prove their point that inspections are not required. We can only hope that it will not be the case.

Currently, software development has been hit hard in the technology sectors, which are early contributors to the current economic downturn. Inspections may be low tech, but they represent a sound investment to guarantee that products released by software companies operate as advertised.◆

Reference

1. Radice, Ronald A. High Quality Low Cost Software Inspections. Andover, Mass.: Paradoxicon Publishing, Jan. 2002.

About the Author

Louis A. Poulin is president of GRafP Technologies. He has been involved in assessing the capability of information technology organizations and in developing hazard evaluation, hazard monitoring, and hazard prevention tools and methodologies applicable to various fields. Prior to this, Poulin served in the Canadian Navy as a combat systems engineering officer. He is a member of the Institute of Electrical and Electronics Engineers and a fellow of the Engineering Institute of Canada. Poulin has a bachelor's degree in engineering physics, a certificate in naval engineering, and a master's degree in electrical engineering.

550 Sherbrooke St. West
Suite 777
Montreal, Quebec
Canada H3A 1B9
Phone: (514) 847-0900
E-mail: lpoulin@grafp.com

Application of Lightweight Formal Methods in Requirements Engineering

Vinu George and Dr. Rayford Vaughn
Mississippi State University

Using formal methods in software development is an important step toward achieving correctness, consistency, and understanding in the software development process. This use of formalism can be informal, semiformal, or formal. An evolving approach known as lightweight formalism uses the application of formal methods that are less "rigorous" than normally expected. This article overviews advantages and disadvantages in the application of formal methods in software development and discusses the lightweight formal approach with emphasis given to the requirements phase of software development.

There is little doubt in the software engineering community that requirements engineering plays an important role in the development of a software system. Requirements engineering [1] is generally divided into five stages: *elicitation* of the requirements from the customer or user, *analysis* of the elicited requirements, *management* of the requirements (i.e., controlling of requirements), *verification* of requirements, and *documentation*.

Requirements engineering can employ a variety of methods that effectively capture requirements. Success with this phase of the development life cycle is considered crucial in that the remainder of lifecycle activities is highly dependent on this early foundation (in terms of schedule, cost, and user acceptance). These methods are broadly divided into three categories: *informal*, *semiformal*, and f*ormal methods* [2], each having its own advantages and disadvantages.

In the requirements phase, requirements are often specified informally with a language in which the customer or end-user is familiar. This often results in the use of natural language (e.g., spoken and written English) to create a requirements document such as the system requirements specification or concept of operations [2]. Such informal specifications are not adequate in that they are often inaccurate, inconsistent, and ambiguous [2]. Natural language specifications are also very lengthy, making them difficult to check for completeness.

Applying semiformal methods that emphasize the use of graphical representations of the software being built can mitigate this disadvantage. A major problem with the semiformal approach, however, is the lack of precise semantics, which may lead to ambiguous interpretation of certain requirements. Sometimes the application of formal methods is offered as a solution to problems associated with both the informal and semiformal approaches.

Due to space constraints, CrossTalk *was not able to publish this article in its entirety. However, it can be viewed in this month's issue on our Web site at <www.stsc.hill.af.mil/crosstalk> along with back issues of* CrossTalk.

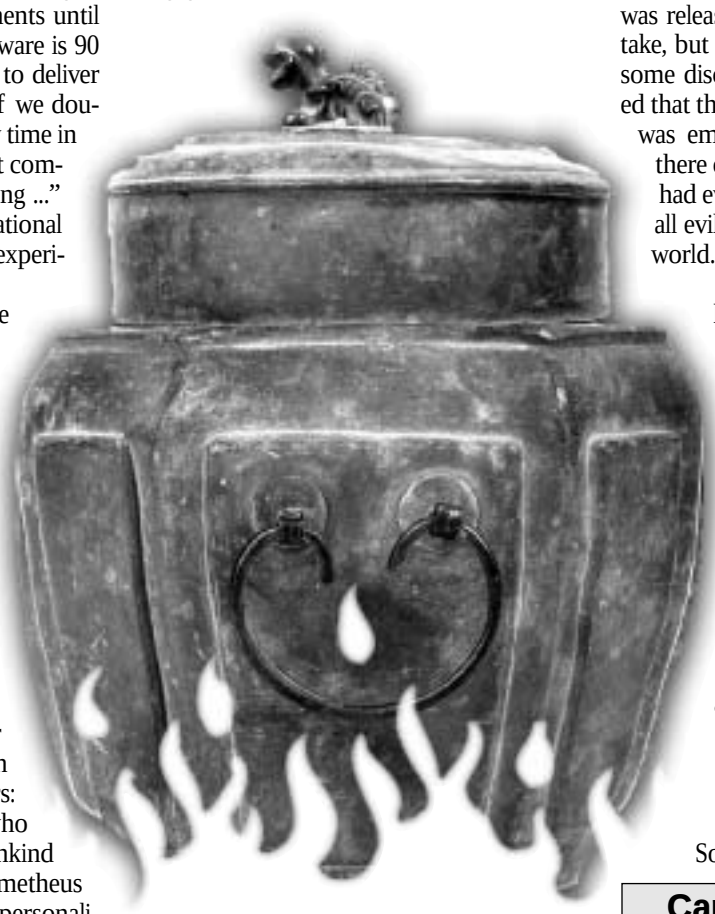
Pandora and the Magic Vase

There are phrases we hear over and over while investigating software developments gone amuck. Some we hear in many developments resemble these: "Our productivity is 25 percent higher than we achieved on the last project because we are working smarter this time." Or, "The schedule is tight, but we have polished the requirements until they are rock solid." Or, "The software is 90 percent COTS. We should be able to deliver the product in six months." Or, "If we double the staff, we can cut the delivery time in half." Or finally, "We are 90 percent complete. Just a few more days of testing ..." You can probably add to this irrational bunch of statements your own experiences.

I am here to declare in true 1990s fashion that these absurdities are not our fault. No, I am not going to blame them on our parents, our schools, the environment where we grew to adulthood, or the occasional use of prescription drugs. These statements come from our genes, and there is nothing we can do to restrain ourselves from uttering them. Let me explain.

The source of this weakness has been with us since creation, at least according to tradition. Our story begins a long, long time ago in ancient Greece with two brothers: Prometheus and Epimetheus, who were among the titans prior to mankind being established on Earth. Prometheus was a young intelligent god whose personality and disposition would have made him an extreme sports star today. He was tasked by Zeus to create mankind. His brother, Epimetheus, was simpler (much more) than his older brother, and much less adventurous. All of his friends called him Epi the Lesser. Epimetheus was tasked to create the other creatures.

One day Prometheus tricked Zeus and stole divine fire from Mount Olympus for the benefit of mankind. Zeus, the chief god, was irate about the theft of this one thing that belonged only to the gods. He went ballistic and swore that Prometheus and all mortals would suffer for this affront. Zeus ordered Hephaestus to create Pandora from earth and water as vengeance upon man and his benefactor, Prometheus. The gods endowed the beautiful and seductive damsel (of course) with every charm including curiosity and deceit. Zeus reasoned that Prometheus was smart and would probably see through his plan, so he arranged for Pandora to marry Epi who, as I said before, was not the sharpest rock in the box. Zeus gave her a vase (no, it wasn't a box) as a wedding gift. Prometheus had warned his slow-witted brother Epi about accepting gifts from Zeus, but Epi was dazzled by Pandora's beauty.



There are two versions of what transpired at this point. Version one says that Zeus forbade her to open the vase. Despite Prometheus' warnings, Epi allowed Pandora's womanly curiosity to win out and she opened the vase. Version two says that Pandora's task was to con Epi into opening the vase. Again, in spite of Prometheus' warnings, Pandora was so beautiful and, literally, irresistible, Epi could not refuse. After a brief casting of her wiles, Pandora led poor Epi into taking the lid off the vase.

When the lid was lifted (who did the deed is a religious issue), there was a tremendous whoosh and all but one of the evils, sorrows, and diseases were released to forever plague mankind. Evils like hard work, greed, lust, envy, television, Eminem, and Britney Spears escaped and spread over the entire earth.

Epimetheus may not have been too sharp, but he was fast. He managed to close the vase before the worst of all afflictions was released. Epi felt terrible about his mistake, but Zeus' work was not finished. After some discussion, Epi and Pandora concluded that the worst had been done and the vase was empty. Epi slowly lifted the lid, and there came a scream as hideous as anyone had ever heard, even on TV. The worst of all evils had finally been unleashed on the world. That evil was ... HOPE.

Because of hope we can never learn from our mistakes. History can never be our teacher. We will forever be cursed with unbounded optimism. Oddly, software program managers and developers seem to have been more receptive to this evil than any other mortal. What was it I heard yesterday? "If you adopt this (tool, technique, programming language, etc.), your productivity will improve an order of magnitude, and you will never again make an error!"

Sure it will!

— Randy Jensen
randall.jensen@hill.af.mil
Software Technology Support Center

Can You BackTalk?

Here is your chance to make your point, even if it is a bit tongue-in-cheek, without your boss censoring your writing. In addition to accepting articles that relate to software engineering for publication in CrossTalk, we also accept articles for the BackTalk column. BackTalk articles should provide a concise, clever, humorous, and insightful article on the software engineering profession or industry or a portion of it. Your BackTalk article should be entertaining and clever or original in concept, design, or delivery. The length should not exceed 750 words.

For a complete author's packet detailing how to submit your BackTalk article, visit our Web site at <www.stsc.hill.af.mil>.

Get Your Requirements Right the First Time

The Requirement for Good Requirements will be offered **March 11-13** in the vicinity of Hill AFB, UT and again on **April 22-24** in the vicinity of Hanscom AFB, MA. This seminar covers the fundamentals of requirements engineering, analysis, elicitation, documentation, and verification and validation. STSC consultants will utilize their many years of hands-on experience to show attendees how to get their requirements right the first time. The seminar also includes planned exercises to help participants solidify the concepts they learn.

The 2003 STSC Seminar Series

January 14-16	Software Project Management	Hill AFB Vicinity
February 18-20	Software Project Management	Hanscom AFB Vicinity
March 11-13	The Requirement for Good Requirements	Hill AFB Vicinity
April 22-24	The Requirement for Good Requirements	Hanscom AFB Vicinity
May 13-15	Software Schedule and Cost Estimation	Hill AFB Vicinity
June 17-19	Introduction to CMMI	Hanscom AFB Vicinity
July 15-17	Introduction to CMMI	Hill AFB Vicinity
August 19-21	The Risks of Not Being Risk Conscious: Software Risk Management Basics	Hill AFB Vicinity
September 16-18	Software Quality Assurance	Hill AFB Vicinity
October 14-16	Why Is Buying Software So Difficult?	Hill AFB Vicinity
November 18-19	Bringing it All Together for the Software Manager (Software Best Practices: An Executive's Perspective)	Hill AFB Vicinity

These seminars are **FREE** to U.S. government employees, however seating is limited, so act quickly.

For additional information, visit our Web site at www.stsc.hill.af.mil

SPACE IS LIMITED. To reserve your place at any of these seminars, contact Debra Ascuena at 801-775-5778 (DSN 775-5778) or debra.ascuena@hill.af.mil.

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

Published by the Software Technology Support Center (STSC)

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

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