Configuration Management Fundamentals

Software Technology Support Center

The U.S. Air Force's Software Technology Support Center offers an updated and condensed version of the "Guidelines for Successful Acquisition and Management of Software-Intensive Systems" (GSAM) on its Web site <www.stsc.hill.af.mil/resources/tech_docs>. This article is taken from Chapter 9 "Configuration Management" of the GSAM (Ver. 4.0). We are pleased that all editions have been so well received and that many individuals and programs have worked hard to implement the principles contained therein. The latest edition provides a usable desk reference that gives a brief but effective overview of important software acquisition and development topics, provides checklists for rapid self-inspection, and provides pointers to additional information on the topics covered.

Change is a constant feature of software development. To eliminate change is to remove the opportunities to take advantage of lessons learned, to incorporate advancing technology, and to better accommodate a changing environment. Refusal to incorporate change can mean system limitations and early obsolescence, which, in the world of technology, can sign your system's death certificate before it is born. However, change is not universally benign and must be controlled in its introduction to a project.

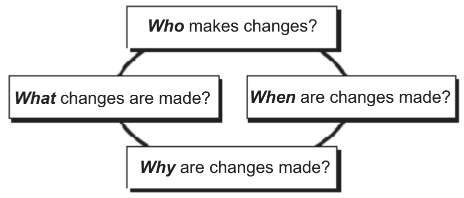
All projects change something. As a project is executed, changes to the initial project plan and products are a natural occurrence. The following are common sources of changes:

- Requirements. The longer the delivery cycle, the more likely they will change.
- Changes in funding.
- Technology advancements.
- Solutions to problems.
- Scheduling constraints.
- Customer expectations.
- Serendipitous (unexpected) opportunities for an improved system.

Some of these changes may appear as options while others may be mandated from above or by circumstance, as in the loss of funding. While all progress is accompanied by change, not all change is indicative of progress. If not properly handled, change can slip the schedule, affect the quality, and even kill the project. As a project draws closer to its completion, the impacts of change are more severe [1]. Clearly, a mechanism is needed to control change.

In software development and other projects, proposed changes must be evaluated to determine their overall contribution to the project goals. Do they lead to improvements or do they ultimately impede or lower project quality? Even those changes that are ultimately beneficial must be controlled in their introduction and implementation.

Figure 1: *Configuration Management Questions*


Who makes changes?
What changes are made?
When are changes made?
Why are changes made?

Putting a bigger engine in a plane may improve its capabilities, but it cannot be implemented until the aircraft's structure has been found capable or been upgraded to support the increased weight and thrust.

Configuration management (CM) is the process of controlling and documenting change to a developing system. It is part of the overall change management approach. As the size of an effort increases, so does the necessity of implementing effective CM. It allows large teams to work together in a stable environment while still providing the flexibility required for creative work [2]. CM in a software environment is an absolute necessity. CM has three major purposes [1]:

1. Identify the configuration of the product at various points in time.
2. Systematically control changes to the configuration.
3. Maintain the integrity and traceability of the configuration throughout the product life cycle.

CM accomplishes these purposes by answering and recording the answers to the change questions: who, what, when, and why, shown in Figure 1 [1]. Being able to answer these questions is a sign of effective CM.

Effective CM provides the following essential benefits to a project:

1. Reduces confusion and establishes order.
2. Organizes the activities necessary to maintain product integrity.
3. Ensures correct product configurations.
4. Limits legal liability by providing a record of actions.
5. Reduces life-cycle costs.
6. Enables consistent conformance with requirements.
7. Provides a stable working environment.
8. Enhances compliance with standards.
9. Enhances status accounting.

In short, CM can provide cost effective project insurance when properly planned, organized, and implemented. It must be integral to your overall project execution, and to your charter/customer agreement. Proposed changes must be dealt with systematically, promptly, and honestly [1]. If the CM process is unreasonable or unresponsive, people will try to circumvent the process, leading to chaos and a loss of the benefits of true CM.

Process Description

While CM is a major element of a change control program, it is such a multifaceted discipline that it should be considered not simply as another activity, but as a program in and of itself. Establishing an effective CM program requires an understanding of CM functions and of the overall CM process.

Functions of Configuration Management

CM is comprised of four primary functions: identification, change control, status accounting, and auditing. These are shown in Figure 2, along with their subfunctions. All CM activity falls within the bounds of these functions.

Identification

This function identifies those items whose configuration needs to be controlled, usually consisting of hardware, software, and documentation. These items would probably include such things as specifications, designs, data, documents, drawings, software code and executables, components of the software engineering environment (compilers, linkers, loaders, hardware environment, etc.), and hardware components and assemblies. Project plans and guiding documents should also be included, especially the project requirements. A schema of names and numbers is developed for accurately identifying products and their configuration or version level. This must be done in accordance with project identification requirements. Finally, a baseline configuration is established for all configuration items and systems. Any changes to the baseline must be with the concurrence of the configuration control organization [2].

Although key components to be managed are requirements and source code, related documentation and data should be identified and placed under CM control. It is important to store and track all environment information and support tools used throughout the software life cycle to ensure that the software can be reproduced. Table 1 lists examples of items typically put under CM control.

Change Control

Configuration control establishes procedures for proposing or requesting changes, evaluating those changes for desirability, obtaining authorization for changes, publishing and tracking changes, and implementing changes. This function also identifies the people and organizations who have authority to make changes at various levels (configuration item, assembly, system, project, etc.,) and those who make up the configuration control board(s) (CCB). (According to IEEE 610.12 [3], a CCB is a group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes.)

Additionally, various change criteria are defined as guidelines for the control organizations. Different types of configuration items or different systems will probably need different control procedures and involve different people. For example, software configuration control has different needs and involves different people than communications configuration control and would probably require different control rules and a different control board [2]. Configuration change control activities include the following:

- Defining the change process.
- Establishing change control policies and procedures.
- Maintaining baselines.
- Processing changes.
- Developing change report forms.
- Controlling release of the product.

A generic software change process is identified in Figure 3 (see next page).

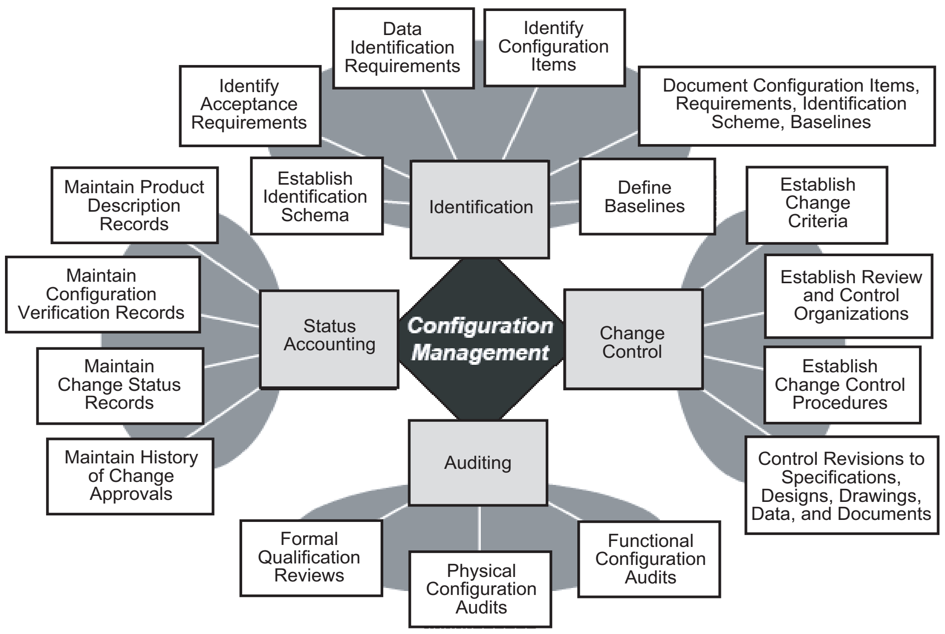

Data Identification Requirements
Identify Configuration Items
Identify Acceptance Requirements
Document Configuration Items, Requirements, Identification Scheme, Baselines
Maintain Product Description Records
Establish Identification Schema
Identification
Define Baselines
Establish Change Criteria
Maintain Configuration Verification Records
Status Accounting
Configuration Management
Change Control
Establish Review and Control Organizations
Maintain Change Status Records
Establish Change Control Procedures
Maintain History of Change Approvals
Auditing
Control Revisions to Specifications, Designs, Drawings, Data, and Documents
Formal Qualification Reviews
Physical Configuration Audits
Functional Configuration Audits

Figure 2: *Major Functions of Configuration Management [2]*

Status Accounting

Status accounting is the documentation function of CM. Its primary purpose is to maintain formal records of established configurations and make regular reports of configuration status. These records should accurately describe the product, and are used to verify the configuration of the system for testing, delivery, and other activities. Status accounting also maintains a history of change requests and authorizations, along with status of all approved changes. This includes the answers to the CM questions in Figure 1 [2].

Key information about the project and configuration items can be communicated to project members through status accounting. Software engineers can see what fixes or files were included in which baseline. Project managers can track completion of problem reports and various other maintenance activities. Minimal reports to be completed include transaction log, change log, and item *delta* report. Other typically common reports include resource usage, *stock status* (status of all configuration items), changes in process, and agreed-upon deviations [6].

Auditing

Effective CM requires regular evaluation of the configuration. This is done through the auditing function, where the physical and functional configurations are compared to the documented configuration. The purpose of auditing is to maintain the integrity of the baseline and release configurations for all controlled products [2]. Auditing is accomplished via both informal monitoring and formal reviews.

Configuration auditing verifies that the software product is built according to the requirements, standards, or contractual agreement. Test reports and documentation are used to verify that the software meets the stated requirements. The goal of a configuration audit is to verify that all software products have been produced, correctly identified and described, and

Table 1: *Items Under CM Control*

Items Under CM Control	
System data files	Source code modules
Requirements specifications	System build files/scripts
Design specifications	Interface specifications
Test plans	Software architecture specifications
Test data sets	Test procedures
User documentation	Test results
Quality plans	Software development plan
Compilers	Configuration management plans
Debuggers	Linkers and loaders
Shell scripts	Operating systems
Other related support tools	Third-party tools
Development procedures and standards [4]	Procedure language descriptions

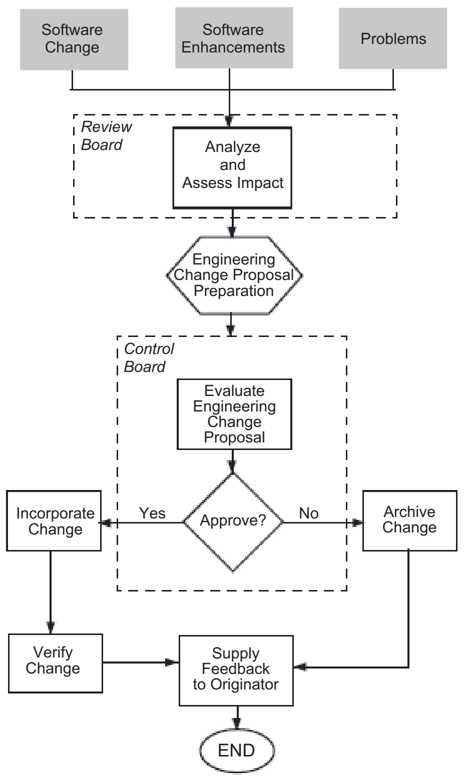


Figure 3: *Generic Change Process* [5]

that all change requests have been resolved according to established CM processes and procedures. Informal audits are conducted at key phases of the software life cycle.

There are two types of formal audits that are conducted before the software is delivered to the customer: Functional Configuration Audit (FCA) and Physical Configuration Audit (PCA). FCA verifies that the software satisfies the software requirements stated in the System Requirements Specification and the Interface Requirements Specification. In other words, the FCA allows you to validate the system against the requirements. The PCA determines whether the design and reference documents represent the software that was built. Configuration audit answers the questions, "Does the system satisfy the requirements?" "Are all changes incorporated in this version?" Configuration audit activities include the following:

- Defining audit schedule and procedures.
- Identifying who will perform the audits.
- Performing audits on established baselines.
- Generating audit reports.

Establishing a Software Baseline Library

In support of the above activities, a software baseline library is established. The library is the heart of the CM system. It serves as the repository for the work products created during the software life cycle. Changes to baselines, and the release of software products, are systematically controlled via the change control and configuration auditing functions. The software library provides the following:

- Supports multiple control levels of Software Configuration Management (SCM).
- Provides for the storage and retrieval of configuration items or units.
- Provides for the sharing and transfer of configuration items or units between control levels within the library.
- Provides for the storage and recovery of archive versions of configuration items or units.
- Helps to ensure correct creation of products from the software baseline library.
- Provides storage, update, and retrieval of CM records.
- Supports production of CM reports.
- Provides for maintenance of library structure [7].

In the past, libraries have been composed of documentation on hard copy and software on machine-readable media. Today, with the advances in information technology and standards that encourage contractors to use automated processing and electronic submittal techniques, organizations are moving toward maintaining all information on machine-readable media.

Configuration Management Process

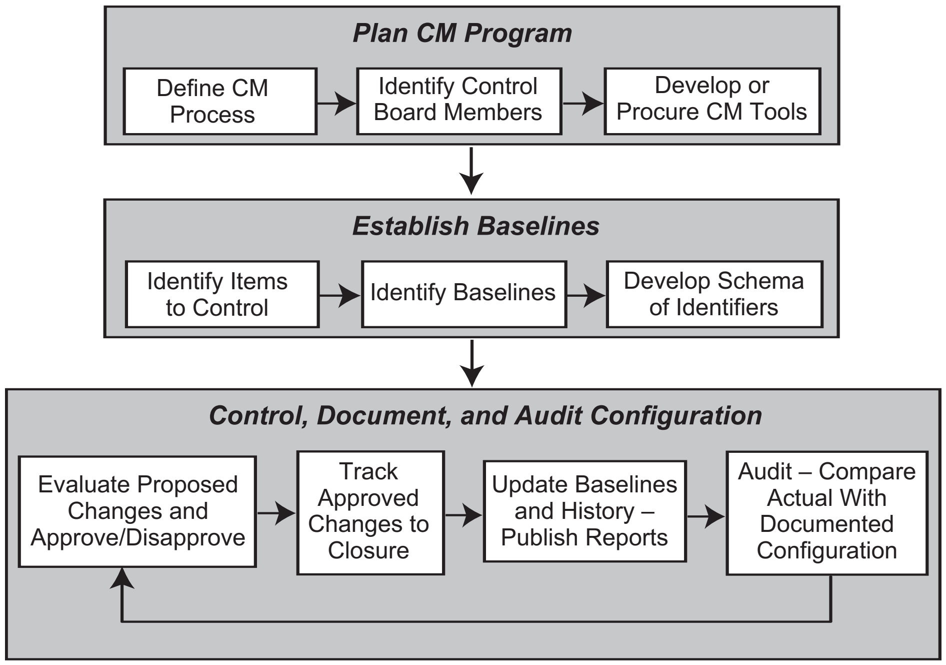
Understanding what CM is supposed to accomplish is one thing. Putting it into practice is another. As with most project activities, CM begins with planning. With a plan, configuration baselines can be established. Following this initial configuration identification, the cyclical configuration control process is put into motion. These three major CM implementation activities are shown in Figure 4.

Planning

Planning begins by defining the CM process and establishing procedures for controlling and documenting change. A crucial action is the designation of members of the CCB. Members should be chosen who are directly or indirectly involved or affected by changes in configuration. For example, a software CCB would obviously be populated with representatives from different software teams, but software affects many more aspects of a project. There should also be representatives from the hardware, test, systems, security, and quality groups as well as representatives from project management and possible other organizations.

Not all changes would be reviewed by this august body. Changes occur at different system levels and affect different portions of the overall system. Many changes will probably only affect a small subset of the system and could therefore be reviewed and approved by a smaller group. Some sort of delineation of change levels should be made during planning to keep change decisions at the proper level. While software CM is essen-

Figure 4: *Configuration Management Implementation Process*


Plan CM Program
Define CM Process
Identify Control Board Members
Develop or Procure CM Tools
Establish Baselines
Identify Items to Control
Identify Baselines
Develop Schema of Identifiers
Control, Document, and Audit Configuration
Evaluate Proposed Changes and Approve/Disapprove
Track Approved Changes to Closure
Update Baselines and History – Publish Reports
Audit – Compare Actual With Documented Configuration

tial, there may need to be other CCBs to control change in other areas of the project. Again, this is something that should be worked out in the planning phase.

Various software tools exist that can facilitate the CM process flow and maintain configuration history. Using a CM software tool is highly recommended. The temptation will be to choose a tool because it looks good in a demonstration and then build the CM process around it. The process should be defined first, and then a tool chosen to facilitate the process.

Establishing Baselines

Once the CM program exists on paper, it must be determined just what configurations it will control. The second major step of implementing effective CM is identifying what items, assemblies, code, data, documents, systems, etc. will fall under configuration control. With the configuration items identified, the baseline configuration must be identified for each item. For items that already exist, it may prove to be nothing more than examining or reviewing and then documenting. For those items that have not been developed yet, their configuration exists in the requirements database or in the project plans. Until they come into physical or software reality, changes to their configuration will consist only of changes to the requirements or plans.

Another essential activity in this step is developing a schema of numbers, letters, words, etc. to accurately describe the configuration revision, or version, for each general type of configuration item. There may be project requirements that dictate some type of nomenclature, or there may be an organizational or industry standard that can be used as the basis for configuration identification.

Controlling, Documenting, and Auditing

When the baselines have been established, the challenge becomes one of keeping the actual and documented configurations identical. Additionally, these baselines must conform to the configuration specified in the project requirements. This is an iterative process consisting of the four steps shown in Figure 4.

All changes to the configuration are reviewed and evaluated by the appropriate configuration control representatives specified in the CM plan. The change is either approved or disapproved. Both approvals and disapprovals are documented in the CM history. Approved changes are published and tracked or monitored until they are implemented. The appropriate configuration baseline is then updated, along with all other applicable documents, and reports are published and sent to affected organizations indicating the changes that have occurred. At selected time intervals and whenever there appears to be a need, products and records are audited to ensure the following:

- The actual configuration matches the documented configuration.
- The configuration is in conformance with project requirements.
- Records of all change activity are complete and up-to-date.

The controlling, documenting, auditing cycle is repeated throughout the project until its completion.

Updating the CM Process

It is unlikely a perfect CM program will be assembled during the initial planning stage. There will be learning and changes in the program that indicate a need for adjustments in the CM process. These may be any mixture of modifications to make it more efficient, responsive, or accurate. When changes in the CM process are needed, consider them as you would any other changes, get the approval of all participating organizations, and implement them as appropriate. It would be ironic indeed to have an unchanging change process.

Configuration Management Checklist

This checklist is provided to assist you in establishing an effective CM program. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

CM Planning

- ❑ Have you planned and documented a configuration management process?
- ❑ Have you identified CCB members for each needed control board?
- ❑ Has CM software been chosen to facilitate your CM process?

Establishing Baselines

- ❑ Have all configuration items been identified?
- ❑ Have baselines been established for all configuration items?
- ❑ Has a descriptive schema been developed to accurately identify configuration items and changes to their configuration?

Controlling, Documenting, and Auditing

- ❑ Is there a formal process for documenting and submitting proposed changes?
- ❑ Is the CCB active and responsible in evaluating and approving changes?
- ❑ Is there a *higher authority* to appeal to when the CCB gets *hung*, and cannot come to a consensus?
- ❑ Are all changes tracked until they are fully implemented?
- ❑ Are all changes fully documented in the baseline documents and change histories?
- ❑ Are regular reports and configuration updates published and distributed to interested organizations?
- ❑ Are regular audits and reviews performed to evaluate configuration integrity?
- ❑ Are configuration errors dealt with in an efficient and timely manner?

Updating the Process

- ❑ Is the CM program itself – its efficiency, responsiveness, and accuracy – evaluated regularly?
- ❑ Is the CM program modified to include recommended improvements when needed?

Case Studies

The following case studies outline specific instances where organizations successfully implemented software CM (SCM).

Selecting a CM Tool

At a large aerospace corporation in the Southeast, the CM manager turned in a recommendation to purchase a CM automated tool that would satisfy all requirements identified by the CM groups. Management delayed acting on the recommendation to give the other engineering departments time to review the recommended tool.

In the end, the recommendation to purchase the tool was cancelled. It was felt that while the tool did support the CM organization, it did not adequately address other developmental considerations that the engineering ranks felt were important. Sometime later a different tool was purchased, one that satisfied all the major requirements of SCM, the software developers, SQA, test, integration, and management organizations.

Overcoming Barriers to SCM

During a recent visit to a private-sector corporation (i.e., they did not deal with government contracts) in New England, it was discovered that the developer's major concern about implementing CM activities was *all the restrictions* they would have to deal with. They had been led to believe that CM meant formal controls, restricted access, limited ability to apply creative solutions, and so on. When it was suggested that data can transition to formally controlled baselines through a series of informal control steps, and that CM did not mean a lockdown and bottleneck, they became eager to be involved. After a number of meetings, a phased approach to formal CM allowed for the placement of informal controls and data gathering which led to baselined items. Everyone was pleased with the process.

The developers soon realized they could work together with CM as a team to solve problems rather than as two separate organizations with their own concerns and desired solutions. More importantly, perhaps, the CM group learned that when they got out of their corner office and out onto the engineering floor (being support and service oriented) they quickly became an integral part of the engineering and development process and team.

Implementing CM With an Electronic Database

A team of 35 to 40 developers was developing six computer software configuration items, which all had two or more customer variants as well as maintenance variants. The operating system was Unix, and the development languages were Ada, C, and C++. Implementing classical CM in this type of environment would normally require three to four CM practitioners to handle all the code and document manipulations. The team chose instead to implement a mostly developer-executed CM system called *Effective SCM*. They implemented a Software Query Language-compliant, database driven, process oriented CM system, which supports a rule-based, closed-loop, change-package approach to development.

Daily interaction with the CM system by the developers provided 100 percent tracking and status accounting of everything that happened to any file in the systems without the need for intrusion or interference by CM practitioners. The CM practitioners maintained the process model and performed the configured builds. As a result, CM support to this team was less than one person, and in fact was in the order of 80-120 hours per month instead of the more than 400 hours per month that a classical approach would have used.

The electronic database created by the engineers completely documented the execution of their software development plan. It also tracked the history of every file used in the system including change documents, baselines, and releases for each file. Note that rule-based, closed-loop change control electronically implemented business rules that prevented the creation of a new version without authorization and prevented closure of a change request that had not been implemented.

A change-package approach supports electronic creation of new baselines by application of changes to a previous baseline. The tool electronically adds, replaces, or removes files that are related to the list of changes being made and is very effective in tracking development activities. (Note that *Effective SCM* is an unregistered trademark of BOBEV Consulting. For a complete description, see "Effective Software Configuration Management" in CrossTalk February 1998.)

Lessons Learned

The following are just a sample of the many lessons that have been learned from applying CM and its associated technologies.

The Importance of Planning

With only a few exceptions, if you look at any of the CM standards, manuals, guides, books, etc., you will likely find that CM has four major functions: (1) identification, (2) change control, (3) status accounting, and (4) auditing. In nearly every case, planning is left out. Yet, SCM is using much more complex equipment to establish and maintain complex environments, multiple baselines, multiple environments on multiple platforms, etc. Like everyone else, CM has to do all that faster, cheaper, smarter, and better than before. Planning has become more important than ever.

Planning cannot be interpreted as meaning *a CM Plan has been written*. That is certainly a good start, but much more is needed than just a document that explains SCM's roles and responsibilities. CM planning activities must also include, to name only a few, such things as the following:

- **Metrics.** How long? How many artifacts? When were they created? When were they updated? Where are they?
- **Skill Mix.** What is needed and who has it or who can get it?
- **Infrastructure.** Who is doing what, where, when, and how?
- **Contingencies.** If this happens, then what?
- **Effort Tracking.** Manpower levels.
- **Subcontracts.** Who has responsibility and authority?
- **Resources.** Budget, tool licenses, training, and head count.
- **Matrix Management.** Decentralized work force.
- **Control Transitions.** Informal to formal to field.
- **Records Retention.** What gets kept where and for how long?
- **Control.** Who controls what and how do they do it?
- **Process.** Standardized procedures for repeatability.

Things to Remember

The most significant lessons are the following:

- Get an inside person on your side – an internal champion. They will become an evangelist for your solution to their co-workers.
- Get management buy-in and sponsorship. Management must really want it, not just go along with it. All levels of management need to support SCM. While implementing SCM, keep a focus on management sponsorship at all times.
- Maintain a sense of humor.
- Be flexible and sensitive to corporate culture.
- Seek out the early success.
- Do not use a critical project as pilot.
- Use a systems approach: Where am I? Where do I want to go? How am I going to get there?
- Success is more likely with lots of preparation, focus on CM and developer needs, breadth of participation, online access to sample process and planning templates, and standard terminology.
- Keep it simple. If it is too complex, or gets in the way, it will not get used.
- Communicate, communicate, communicate.◆

References

1. Software Technology Support Center. "Life Cycle Software Project Management." STSC Seminar, 9 Oct. 2001.
2. Software Program Managers Network. "Little Book of Configuration Management." Software Program Managers Network, Nov. 1998 <www.spmn.com/products.html>.
3. Institute of Electrical and Electronics Engineers. "Standard 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology." New York: IEEE, 1990.
4. Kasse, Tim. Software Configuration Management for Project Leaders. Proc. of Software Technology Conference, Apr. 1997.
5. Berblack, Ronald H. Software Configuration Management. John Wiley & Sons, New York, 1992.

6. Ben-Menachem, Mordechai. Software Configuration Management Guidebook. McGraw-Hill, 1994.
7. Olson, Timothy G., et al. "A Software Process Framework for the SEI Capability Maturity Model: Repeatable Level." CMU/SEI-93-TR-7. Pittsburgh, PA: Software Engineering Institute, 1993.

CrossTalk did not have room to cover the fundamentals of testing in this month's theme section, "Configuration Management and Test." The Guidelines for Successful Acquisition and Management of Software-Intensive Systems (GSAM) is also a good source for information on the basics of testing and many other software topics. For more information on test, see Chapter 12 of GSAM Ver.. 4.0 at <http://www.stsc.hill.af.mil/resources/tech_docs/gsam4.html>.

About the Author

The Software Technology Support Center (STSC) produced the "Guidelines for Successful Acquisition and Management of Software-Intensive Systems." Visit the STSC Web site at <www.stsc.hill.af.mil/resources/tech_docs> to access all 17 chapters of this document. The STSC is dedicated to helping the Air Force and other U.S. government organizations improve their capability to buy and build software better. The STSC provides hands-on assistance in adopting effective technologies for software-intensive systems. The STSC helps organizations identify, evaluate, and adopt technologies that improve software product quality, production efficiency, and predictability. Technology is used in its broadest sense to include processes, methods, techniques, and tools that enhance human capability. The STSC offers consulting services for software process improvement, software technology adoption, and software technology evaluation, including the Capability Maturity Model® Integration℠, software acquisition, project management, risk management, cost and schedule estimation, configuration management, software measurement, and more.

Software Technology
Support Center
6022 Fir AVE BLDG 1238
Hill AFB, UT 84056-5820
Phone: (801) 586-0154
DSN: 586-0154
E-mail: stsc.consulting@hill.af.mil

Web Sites

Configuration Management Yellow Pages
www.cmcrossroads.com/yp/index.php?oldpage=configuration_management.html
The Configuration Management Yellow Pages is a categorized database of links to configuration management and development resources. The site, originally created by Andre van der Hoek, is now hosted at CM Crossroads in a format that allows any member to add or update a new resource and to review and rate existing ones.

Test and Measurement World
www.tmworld.com
This is the online version of *Test & Measurement World* and *Test & Measurement Europe*, which cover the electronics testing industry, providing how-to information for engineers who test, measure, and inspect electronic devices, components, and systems.

Software Configuration Management
www.sei.cmu.edu/legacy/scm
This is the Software Configuration Management area of the Software Engineering Institute's (SEI) Web site. This area is intended to share the configuration management research done by the SEI between 1988 and 1994 and to provide pointers to other useful sources of information on Software Configuration Management.

Software Testing Institute
www.softwaretestinginstitute.com
The Software Testing Institute (STI) provides industry publications, research, and online services, including a software testing discussion forum, the STI Resource Guide, the Automated Testing Handbook, STI Buyer's Guide, and more.

Data Interchange Standards Association
http://www.disa.org
Home to numerous organizations developing e-business standards, the Data Interchange Standards Association helps individuals and the business community improve business processes, reduce costs, increase productivity, and take advantage of new opportunities.

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.

Institute of Electrical and Electronics Engineers
www.ieee.org
The IEEE promotes the engineering process of creating, developing, integrating, sharing, and applying knowledge about electrical and information technologies and sciences. IEEE provides technical publications, conferences, career development assistance, financial services and more.

Practical Software and Systems Measurement
www.psmsc.com
Practical Software and Systems Measurement (PSM): A Foundation for Objective Project Management was developed to meet today's software and system technical and management challenges. The Department of Defense and the U.S. Army sponsor PSM. The goal of the project is to provide project managers with the objective information needed to successfully meet cost, schedule, and technical objectives on programs.