Who Needs John Wayne ?

The Software Technology Support Center (STSC) Web site gets more than 80,000 hits per month, most of which are on *CROSSTALK* articles, past and present. Often, the most popular article subject is configuration management (CM). The STSC Web site also has pages dedicated to CM resources, and CM receives nearly double the number of hits of any other technology area. Professionals are obviously looking for answers to CM questions they face every day. Ironically, the CM consultation service we offer to defense organizations is the least requested of all our services. I can conclude one of two things from this: CM is not glamorous enough to attract the funding it needs for implementing improvements, or CM managers are all rugged individualists. The latter, if true, can probably be explained in terms of our cultural mind-set.

We seem to be heavily influenced by the John Wayne rugged individualist approach to life. We struggle to be towers of individual strength and isolation. To a large degree, we work alone, face crisis alone, and grieve alone. Nowhere is this more clearly demonstrated than in the way we learn. We read alone, review alone, solve problems alone, and create alone. We pursue this individualistic drive in the face of overwhelming evidence that teams are more productive than individuals because the whole is better than the sum of its parts. Study groups in school find solutions to problems more accurately and in less time and increase the rate of learning.

Our rugged individualist approach to learning is also prevalent in our work habits. How many of us, for example, got together as a group to learn the latest version of Windows? It is more likely that we pursued a frustrating path of learning it on our own, on the job, and as we needed it and only learned what was immediately required to be semi-functional. What an inefficient way to learn. Educators have long understood that we remember 10 percent of what we read, 20 percent of what we hear, 50 percent of what someone coaches us through, and 90 percent of what we do. Reading this publication, for example, is only the beginning of the process you must follow to assimilate and effectively apply the CM principals and concepts introduced here.

To internalize any new concept, you must first go through a discovery process of contact and awareness followed by an understanding of what is to be gained, followed by evaluation and trial usage. Finally, after much effort, the new concepts are mastered, adopted, and institutionalized. CM is no different. New habits must be formed to effectively apply CM principles. This is a difficult process even when we are committed to the change; a halfhearted effort will take much longer. Because organizations have many people in varying degrees of resistance to change (who by definition are halfhearted), it becomes even more imperative to use a tutor or consultant to stimulate organizational change. To rely on individualism to train a team in CM techniques may satisfy your inner John Wayne, but it will not get the job done. ◆

Letter to the Editor

Outsourcing's Hidden Costs Deserve Closer Inspection

I read "Outsourcing and Privatizing Information Technology – Re-examining the Savings," *January 1999*, with particular interest. As a former Marine Corps comptroller now heavily involved in information technology (IT), I have seen this issue from both sides. Although there is real pain in forcibly pushing our high-technology public servants from public to private industry, the real issue is finding the best value (in the long run) for the taxpayer. If outsourcing can lower costs, it is worth examining.

Michael Brower's article contains contradictory points. If government cannot compete with the high-wage private sector for workers, how does outsourcing to that sector save money? Yet, Mr. Brower argues that government workers have to worry about low-cost civilian jobs. Juxtaposing the two arguments, it is clear that one is wrong. You cannot have it both ways.

Still, I agree with Mr. Brower's main point. Office of Management and Budget Circular A-76-style outsourcing tends toward failure in IT. An A-76 study should determine whether maintaining government IT resources is more costly than maintaining effective management and control of contracted resources. Traditional A-76 clearly quantifies the former. The latter is usually severely underestimated. So, even high-performance, high-quality government organizations risk being dismantled. The government then finds that it must hire another set of contractors to supervise the first set because too much in-house expertise is gone. Sometimes, even a third set of contractors is hired to manage the second set. Such recursively determined additional cost is seldom budgeted. A-76 "savings" disappear and so does effective mission performance. (These opinions are mine and should not be attributed to my employer or to any government agency.)

Gary A. Ham
Woodbridge, Va.

CCB – An Acronym for "Chocolate Chip Brownies"?
A Tutorial on Control Boards

Reed Sorensen
Software Technology Support Center

This article reviews the basic concepts of control boards. It answers the questions, What is a change board? When is a change board needed? How is a change board established and run? What is the role of a change board in the software development lifecycle?

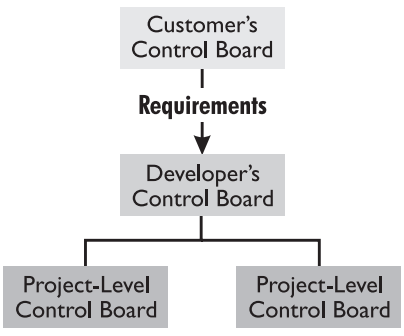
HAVE YOU NOTICED the variety of names given that institution known as the CCB? Configuration Control Board, Change Review Board, Change Implementation Board, and Software Configuration Control Board are popular names. It does not take a great deal of imagination to come up with other possible definitions for CCB. The lack of a universal definition contributes to the confusion that sometime surrounds this critical part of any serious attempt at software configuration management (CM).

Control Board Defined

For the purposes of this article, the term "control board" will refer to a body that provides the means to implement change control at optimum levels of authority. This hierarchical approach is shown in Figure 1.

As shown in Table 1, there are two types of change board: Those that make business decisions and those that make technical decisions. In light of these distinctions, the myriad names mentioned in the first paragraph need to be further examined. Table 2 shows that to know the name of a change board is not enough to know what type of board it is. An "SCCB" may be a business decision change board or a technical decision change board, depending on the organization that chose the name.

Figure 1. *Levels of authority.*



Example Scenario

The two types of boards work together. Consider, for example, how a change request to ensure that system XYZ is year 2000 (Y2K)-compliant would be processed.

The business decision change board authorizes someone to do a preliminary analysis that includes a rough order-of-magnitude cost estimate to implement the change as well as a finding on its technical feasibility. Based on the preliminary analysis, that same change board considers the risks and benefits of implementing, deferring, or ignoring the change request. They consider implementation cost, available resources, and political implications.

If the business decision control board decides to proceed with implementation of the change, a project is initiated to do so. As the project proceeds, Y2K issues and proposed solutions are documented as change requests. These change requests generally do not need the consideration of the business decision control board but rather the technical decision control board, which deals with issues such as how many bytes to use and what type of date representation is appropriate. The sidebar "Sample Control Board Meeting Discussions" on the next page compares conversations in each type of board meeting.

Business Decisions	• Cost. • Schedule. • Function for the whole system.
Technical Decisions	• Specific schedules for partial functions. • Interim delivery dates. • Common data structures. • Design changes.

Table 1. *Decision types [1].*

When to Establish Control Boards

As a project manager of software developers, how do you know if you need to establish control boards? You can answer this question by determining if the manager's near-term project issues in the following example sound familiar. (These issues pertain to a system that has been released and used by customers for six weeks.)

- Get all problems documented in the problem-tracking tool (all problems need to be identified and described).
- Deal with George having quit last week. He was the technical lead for design to deal with technology issues of implementation (Sybase, printing, etc.) Bill and Callie also have quit from the stress of producing almost daily releases in response to customer change requests. There is no documentation of the things George did, so he left a big hole in our staff.

	Institute of Electrical and Electronics Engineers STD 1042-1987	Software Engineering Institute Capability Maturity Model	Institute of Configuration Management [2]	Anonymous software development organization
Boards that make *business* decisions regarding proposed changes.	CCB (Configuration Control Board)	SCCB (Software Configuration Control Board)	CRB (Change Review Board)	SCCB (Software Configuration Control Board)
Boards that make *technical* decisions regarding proposed changes.	SCCB (Software Configuration Control Board)	Software Engineering Group	CIB (Change Implementation Board)	SCRB (Software Change Request Board)

Table 2. *Examples of names for control boards.*

- After seven batch jobs are run successfully at customer sites A and B, configure everything under the version control tool.
- Document and describe all interfaces in the next three months (we have a list of the interfaces).

On a project with functioning control boards, these issues would be less likely to surface because control boards

- Ensure that all problems are documented.
- Authorize releases in a controlled fashion based on schedule and cost considerations.
- Establish controlled baselines.
- Ensure that interfaces are documented and controlled.

Establishing and Running a Control Board

Following are three steps to establish and run a control board.

Write a Charter

The charter should describe the board's objectives, scope, membership, roles and responsibilities of members, reporting and approval process (including standard and emergency changes), meeting frequency, and relationship to other boards. Many organizations have a process to review charters to avoid duplication of effort across control boards. For a sample of a control board charter, E-mail a description of your project to me (address at end of article).

Sample Control Board Meeting Discussions

Business Decision Control Board Meeting

Secretariat: Our first agenda item is EX-01, "MIRS Location Codes Addition."

Chairman: Has anyone not reviewed the data package? (Pause.) This is a major requirements change. We are 18 months into development with established functional and allocated baselines. We are on schedule to field the system in six months. As you know, tank maintenance will begin at least two months before delivery. Sally, what is your position?

Software Project Manager: I agree with the estimate of a three-month schedule slip. This change is clearly out of scope, based on the existing requirements baseline. I support funding this as part of a future release rather than slip the schedule.

User Representative: The problem with incorporating the change in a future release is that the new location data will have to be tracked manually in the interim. Despite that, we would rather have the system delivered on schedule so we can eliminate the workload of manually tracking all the data except the location codes.

Chairman: And Gen. Given has repeatedly stated publically that the system will be available in June. We are going to defer this change for a future release.

Technical Decision Control Board Meeting

Secretariat: The next change request on the agenda is No. 19, named "Restructure Shipments Table to Eliminate Data Redundancy."

Database Representative: I have reviewed the data package and discussed it with the original designer. The Shipments Table as designed produces redundant data. The current design violates basic relational database design principles. This design will cause serious data maintenance problems. The solution outlined in the change request to create separate Shipment and Supplier Tables is correct. We need to make this change.

Graphical User Interface Analyst: The changes to the graphical user interface are manageable. We agree with the estimates in the change request and agree with making the change.

Chairman: When can the change be made, tested, and moved to the development library?

Database Representative: By the fifteenth, if we start on Monday.

Chairman: The change request stands approved. Begin implementation Monday.

Chairman	Secretariat
Follow the charter.	Generate the agenda.
Convene regular meetings.	Distribute the data package.
Prioritize agenda items.	Take and distribute minutes.
Conduct the meeting.	Reserve the meeting room.
If members are not prepared, adjourn the meeting.	
Strive for consensus on change request decisions.[1]	

Table 3. *Responsibilities of the chairman and the secretariat.*

Follow the Charter

The chairman convenes and runs the board. For the business decision control board, the chairman is usually the person who controls the money or other resources; for the technical decision control board, it is the project manager. The chairman and secretariat are the key roles on a control board. Their specific responsibilities are listed in Table 3.

The use of consensus varies between authoritative boards and voting boards. Authoritative boards are governed exclusively by the chairman, who listens to the members and then makes a decision. In a voting board process, changes that fail to receive consensus are deferred possibly many times in an attempt to achieve consensus later. The responsibilities of all members of the board are listed in Table 4.

Reach a Decision

For each change request, the board must approve and assign priority, defer for later consideration and possible inclusion with other changes, refer to a higher authority board, or reject.

Five Principles That Govern Control Boards

I have already discussed the first two principals:

- Business decision control boards make business decisions.
- Technical decision control boards make technical decisions.

The final three principles are as follows:

Use Control Boards Throughout the Lifecycle (Principle 3)

Control boards are required to fully implement these four CM processes throughout the software lifecycle.

- Configuration identification – Identify executables, databases, source files, and procedures to be controlled.
- Configuration change control – Control the identified items so that only authorized changes are made.
- Configuration status accounting – Provide management and practitioners "snapshots" of the state of the identified items and associated change requests.
- Configuration audits – Compare the results with original plans.

Table 4. *Responsibilities of all control board members.*

Before the Meeting
• Review the data package.
• Communicate with other members of the board regarding the change.
At the Meeting
• Represent their organization or group.
• Express and coordinate their organization's or group's viewpoints.

Control the Baseline(s) (Principle 4)

Each board establishes the baseline that corresponds to its authority.

- Functional baseline – Can be established when requirements are agreed on by the customer and the developer, usually at the system design review (SDR). The customer *business decision control board* considers the results of the SDR before authorizing the baseline.
- Allocated baseline – Can be established when requirements have been assigned or allocated to software subsystems, hardware, or manual procedures. The developer *business decision control board* considers the results of the software requirements review before authorizing the baseline.
- Developmental baseline – Can be established during implementation when the *technical decision control board*[2] so authorizes. For example, such baselines can be established at various points during informal testing.
- Product baseline – Can be established when the functional configuration audit and physical configuration audit is complete. The customer and developer business decision control boards consider the results of the audits before authorizing the baseline.

Establish Process for Multiple-Project Decisions (Principle 5)

Refer changes involving multiple projects to a higher board. Consider this example: A technical decision control board considers the following change request.

> The "designator" field in the inventory control system allows for only eight planning shops. One of the user sites of the system has assigned all eight planning shops. They need to add a ninth planning shop. The material shipping system also uses the designator field.

The technical decision control board would be correct to refer the change request to a higher authority board because multiple systems will be impacted by changing the designator field.[3]

Summary

Though they bear many names, there are only two types of control boards: Those that make business decisions and those that make technical decisions. Established control boards have a charter that describes their purpose and procedures. Effective control boards can simplify the role of the software project manager and improve the work environment of the practitioner. To do so, the boards must be used throughout the software lifecycle in conjunction with fundamental CM processes. ◆

About the Author

Reed Sorensen is on the technical staff at TRW and is currently working under contract to the Software Technology Support Center. He has over 20 years experience developing and maintaining software and documentation of embedded and management information systems, providing systems engineering and technical assistance to program offices, and consulting with many Department of Defense organizations regarding their software configuration management and documentation needs. He has published articles in *CROSSTALK* on various software-related subjects.

Software Technology Support Center
OO-ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
Voice: 801-775-5738 DSN 775-5738
Fax: 801-777-8069 DSN 777-8069
E-mail: sorenser@software.hill.af.mil

References

1. IEEE Guide to Software Configuration Management, IEEE-STD-1042-1987.
2. Institute of Configuration Management at Arizona State University.

Notes

1. This means general agreement, not unanimous agreement.
2. If the developmental baseline in question includes changes that affect multiple projects, a higher authority will be needed.
3. The higher board would probably be a business decision control board that could authorize analysis of this interface issue.

Configuration Management Web Sites

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

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

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

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

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

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

CM Bibliography
http://liinwww.ira.uka.de/bibliography/SE/scm.html

A Software Engineering Resource List for CM
http://wwwsel.iit.nrc.ca/favs/CMfavs.html

Other Web Sites of Interest

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

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

Government Electronic Industries Association
http://www.geia.org

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

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

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

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

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

Using CM to Recapture Baselines for Y2K Compliance Efforts

Ronald Starbuck
International Billing Services, Inc.

Time is closing in on the year 2000 (Y2K) and those software organizations without product configuration baselines risk failure to meet this hard, "drop-dead" conversion deadline. The purpose of this article is to provide the basic methods essential to achieve the recapture of a product's software baseline to begin the Y2K conversion.

The foremost challenge facing today's software organizations is to make their products Y2K compliant. Failure to achieve compliance will result in immediate product obsolescence at the new millennium's arrival. At the heart of any successful Y2K conversion is the configuration management (CM) system, which has to be flexible enough to accomplish ongoing maintenance of routine fixes while integrating Y2K conversion code into the product [1].

The keystone of CM is the baseline [2]. Organizations that hastily developed software without CM may now be facing Y2K conversion efforts without product baselines. The lack of reference points in this vital area means that an organization cannot effectively support its software products [3].

Watts Humphrey best expresses the importance of CM to Y2K compliance in "Year 2000 Readiness Checklists." [4] He emphasizes that the lack of CM system capability "will most likely [create] problems that could be severe and unrecoverable." Humphrey implores software developers to implement CM immediately rather than wait until their schedule clears.

This article provides the first step to make software Y2K compliant: a method to recapture baselines of systems built without a CM system in place. The recapture process, ironically, requires that a CM system first be implemented.

CM System Defined

The CM system described by Humphrey constitutes more than just the adoption of high-end automated CM tools. Although tools are an important part of a CM system, they do not compose the entire system; rather, it is based on the integration of basic CM functions—configuration identification, control, status accounting, and audits—into a cohesive process that uses defined procedures, documentation, automated tools, and practitioners. Humphrey observes that "The CM system maintains physical and electronic control of the organization's programs and data." [4] It ensures the integrity and reliability of source code, test data, and documentation over time through use of baselines. These software work products must always be kept current and correct to ensure reliable reproducibility and change.

Effective CM systems always originate from a well-written configuration management plan (CMP) [1]. This plan should be written to provide a clear understanding of how a baseline is created and maintained from the initial identification of components and documentation to their release and subsequent configuration control through a change management process. Details in the CMP should include how audits are performed to maintain the integrity and stability of the software product for further development. When properly executed, the CMP mirrors the organization's software process.

Establish Business Rules

Three vital considerations must be determined by management to set the tone for what is to be done for baseline recapture.

- Consider impact to business revenue. Due regard must be given to financial limitations; otherwise, the sequence and scope of the recapture effort cannot be properly determined.
- Ensure that software products under contract support are thoroughly reviewed. Ascertain what the customer purchased for software maintenance to determine which artifacts (software work products) have to be accounted for in the support effort [2]. The range of possibilities can sweep the entire product spectrum from what is needed to support a full 30-year Department of Defense software-intensive system to a short-term interim application.
- Identify which minimum artifacts are needed to support the products. Identify the product's name, software performance requirements, software component lists, and documentation created for support. As elementary as this seems, as Humphrey points out, if they are not recorded, they are soon forgotten and lost. Even with automated CM tool assistance in the software process, documentation is not necessarily archived with the source code.

These business rules establish the requirements to be used by the Configuration Control Board (CCB) to determine types of products, required support artifacts, and the order of events to recapture lost product baselines.

Configuration Control Board

The CCB is the forum where business rules are put into action. Its function is central to manage and control baseline recapture efforts. Requisite responsibilities are defined in a CCB charter, which would outline the CCB's functions; thus,

- It guides the entire recapture effort from start to finish.
- It must resolve all problems and situations that arise during the effort.
- To be effective, it must have clear authority and scope.
- Its primary decision-making resource is the CM system.
- When needed, it takes into account organizational management considerations for decision making.
- It orchestrates all activities from the beginning to the approval of the baselines for CM establishment.
- It determines which products should be recaptured, the method to be used, and the order in which they should be done.
- It resolves all issues such as the most suitable product name to be used from among numerous documented aliases. (Multiple aliases often occur when a CM system was not originally in place.)

CCB members should have experience in many areas of software engineering, including CM, so that they can adequately advise the CCB chairman on such matters as product history, costs, contractual aspects, market conditions, quality requirements, and other proceedings. The chairman should have the authority to implement decisions, including those that require funding authorization. CM is the CCB's conduit to accomplish its configuring and processing decisions. All relationships and interfaces should be identified as part of the process plan for recapture as defined in the CMP.

Tools

CCB activities require a planning and decision-making tool to identify, track, and record findings revealed for each product. A generic example of one tool is shown in Figure 1. This checklist functions as a CM process action and status tool used to get tasks accomplished and monitor progress. Software organizations that use automated CM tools could have much of this information already available. A few characteristics of the checklist are as follows:

System Name				Recapture	yes no
Approach Used	Contracted Internal	Category		Assessment	
		Essential	Not Essential	Current	Not Current
Software Version					
Component Composition (Attached)					
Compiler					
Version					
Tools					
Version					
Directory Name					
Date					
Requirements Documentation					
Publication Date					
Revision					
Maintenance Documentation					
Publication Date					
Revision					
Associated Documentation					
Publication Date					
Associated Test Documentation					
Publication Date					
Approve Disapprove					
Signed				Date	

Figure 1. *Baseline recapture checklist.*

- It is initiated at the direction of the CCB for each product under consideration.
- It ensures consistent organization and formatting of the recaptured data for CCB consideration.
- It includes fields for identification of the software product's name, software composition by version, documentation by revision, and any tools necessary to build the product.
- The "Approach Used" field identifies whether the recapture effort is for a contracted or internally produced item.
- Each list is given a unique CM control number.

- Those lists for contracted support products must include all contracted items.

The CM group and CCB are responsible for completing two highly important portions of the checklist. The first categorizes artifacts to be recaptured as essential or not essential. If an item is designated not essential (not mandatory but nice to have), the CCB chairman annotates the checklist accordingly, and the CM group establishes and maintains the item in its historical files for traceability. Artifact items checked as essential must be locatable and traceable. If they cannot be found or were never generated, a statement to that fact should be noted on the checklist.

The second important portion on the checklist is termed "Assessment." It specifies whether an artifact is current. The CM group records this evaluation after completing an analysis of the adequacy of the artifact to be released. When all items on the checklist have been assessed, the completed checklist provides all the information needed by the CCB chairman to approve or disapprove the baseline recapture effort. Approval signifies the sanctioning of the checklist's configuration information so that the CM group can establish the product's baseline and maintain all the processed checklists.

After CCB deliberations, the final decision rests with the CCB chairman to give the CM group the task of proceeding with investigation and coordination or of closing the checklist out as disapproved. To obtain information may require exploration of both on-line and backup storage such as tapes, disks, or cassettes, depending on the system legacy. Research of soft- and hard-copy documentation, preferably from original departmental documentation libraries, and one-on-one interviews with original system development and maintenance employees can be extremely useful to recapture the baseline. An intangible benefit from recapture is to uncover documentation that may still be useful to support the product. This previously unknown information should be entered on the checklist and validated regarding its relevance to the current released product. The CCB will rule on its inclusion in the recaptured baseline.

When all research and information gathering is completed for a checklist, the CM group members evaluate it for completeness and write their initials on each item. They must determine whether an item is essential or not essential and current or not current with the released configuration. The completed checklist should include a detailed list of all software components called for in the performance requirements and the design documentation. If the checklist is incomplete, it is returned to the originator for completion. Unknown documents uncovered during research are identified by the CM group and brought to the CCB for resolution. Finally, the CCB must determine the adequacy and currency of recovered artifacts with the released product version. Checklist items not complete or adequately documented may require an impact estimate on cost to bring them into agreement with the released product. If the item was a contract deliverable not maintained since the initial release, a business rule may mandate its recovery. The CCB chairman will make that decision.

Recapture Process

The following is a summary of what, as a minimum, the CM group should do to recapture a baseline foundation to ensure parallel software support.

- Initiate a checklist based on known released products by product name.
- Identify the categories of artifacts for potential identification for baseline recapture and forward them to the CCB chairman for concurrence and direction.
- Follow the chairman's direction and coordinate the investigation. To obtain the information may require exploration of both on-line and backup electronic storage of media, depending on the legacy of the particular system.
- Search through such media types as tapes, cassettes, and disks and review associated hard-copy documentation through various departmental documentation libraries.
- Hold one-on-one interviews (if possible, with original team members). These constituents can provide invaluable insight into what happened and what was produced.
- Review the categories on the form and qualify each item as essential or not essential and current or not current with the released configuration. The checklist should contain list of all software components called for in the performance requirements and the design documentation.

After reviewing the checklists, the CCB returns them to the CM group for their assessment and processing. All artifacts uncovered are then analyzed for their adequacy and currency to the current released product version and provided to the board for reconciliation. Those items that are not current require an impact analysis for the estimated effort and resources to bring them into harmony with the software's released version level. Each item is authorized by the CCB chairman based on business rules, contractual commitments, or other higher-level management or product needs. The following sections provide further insight into what and how the software requirements and version information can be recaptured.

Requirements

It is essential to determine which performance requirements were to be achieved by the software. These specifications are generally produced before the code is designed and written and is the source from which the software design is based and coded. With incremental or evolutionary engineering models, the requirements are successively achieved; therefore, its documentation may be produced in sets. This document or set of documents could be in or part of the Statement of Work or in other standalone documents.

If created, an operational user's manual can include the system's operational requirements. Those mature organizations that incorporate software quality assurance (SQA) elements could have additional independent test plans that identify the functions tested and provide verification of produced arti-

facts. It also is possible that SQA has other records that further amplify, describe, or test the product.

Software Version

With these requirements noted in the checklist, the most complex task is to ascertain the relationships between the software artifacts that compose the released product. This decomposition must go down to the version level of build files and source code. CM tools can enable this with minimal effort. The hierarchy between the various components that compose software products can also make discerning the relationships easier. If tools are not employed, there should be some form of software repository from which software components can be identified for each product. The product's software components and the tools used to build them must be defined according to the version level used for baselining [3]. This will require checking records of system administrators to find what was used for a released product.

CM Status Review

Upon completion of each product's recapture effort, all checklists are returned to the CM group for a status review. The CM group determines readiness of the configuration data to be forwarded to the CCB for final baseline sanctioning or to address the particular need at the time to render a decision on some issue or problem. The CM group sorts the checklists into two types of packages: The first includes those deemed as current and ready for final determination by the CCB for baselining. The second are those that need further resolution by either the originator or the CM group. The checklists are verified by the CM group to ensure that the approach used to obtain the information followed the CCB's direction and that the consistency of the information complies with standardization, accuracy, and format for the recorded items.

Of high concern to the CM group is the inclusion of those items it identifies as essential for baselining, as shown in Figure 1. If not provided, a reason must be attached to the checklist so the CCB can determine the need to re-create missing artifacts. The CM group ensures that each artifact identified has either a version or a revision number for each essential artifact. Any missing revision numbers found by the group are returned to the checklist implementers to obtain them. Those artifacts that are not current have to be acknowledged to the board so that a decision can be made regarding the need for an impact analysis to bring them into conformance.

This task is considerably easier when organizations use a common revision scheme in its version application. There will be those unfortunate instances where no revision status was used. This is quite prevalent in chaotic types of development as identified in the Software Engineering Institute Capability Maturity Model. The CM group also verifies that the tools and the revisions used to create the product were included. When completed, the CM group forwards the package to the board with its recommendations on the adequacy of data recaptured for baselining.

Reconciliation

The CCB reviews all packages and provides recommendations to the CCB chairman for final decision. Products not approved for the baseline could require more information or be rated "disapproved" based on business need. If necessary, the checklist is annotated and returned to the CM group for further processing. When the CCB chairman's signature and date appear on the checklist, it signifies the final decision from which the CM group takes action. For baseline recapture to occur, all product components and documentation not current with released software must be recovered and made current. The decision to proceed may depend on the cost required for the effort. Documentation often is the category most likely to be incomplete, especially when referenced by multiple names or aliases. When checklists are finally sanctioned by the CCB chairman, they are archived into the chosen database or CM tool by the CM group. They may also be stored manually. This act constitutes the re-establishment of the product's baseline for authentication by the SQA group. It completes the recapture and provides the foundation needed to begin Y2K conversion.

Summary

Humphrey best expresses the importance of a CM system to achieve Y2K compliance when he states that the lack of a CM system will most likely create problems, some of which could be severe and unrecoverable. CM tools alone are not the entire CM system but part of it. Without an effective CM system, software organizations are likely to lose programs, misapply fixes, or use the wrong test or test data. Software organizations that sacrificed CM for expediency or other reasons are extremely vulnerable to problems related to Y2K conversion. An incomplete or absent baseline directly affects a software organization's ability to support its software. The only option available when a baseline does not exist is recapture. This article explained what is necessary from both management and the CM process to accomplish the recapture of product baselines—the key to completing conversion to Y2K. ◆

About the Author

Ronald Starbuck is a configuration manager at International Billing Services and is involved in improving customer systems' CM processes and process infrastructure. He has spent more than 20 years in various government positions and agencies such as lead programmer for the Navy's P-3 Orion Weapons Systems Simulator, configuration manager at the Sacramento Army Depot for test program sets, and a software quality assurance representative for the Defense Logistics Agency.

International Billing Services, Inc.
Mail Stop 7050
5220 Robert J. Mathews Parkway
El Dorado Hills, CA 95762-5712
Voice: 916-857-6857
E-mail: Ronald_Starbuck@uscs.com

References

1. Software Engineering Institute, CMU/SEI-93-TR-25, 1993, pp. A-5, A-22, A-45.
2. Starbuck, Ronald, "The Misplaced Backbone of Software Change Management," *Software QA*, Vol. 4. No. 5, 1997, pp. 19-21.
3. Bersoff, Edward, Vilas Henderson, and Stanley Siegal, *Software Configuration Management*, Prentice-Hall, Englewood Cliffs, N.J., 1980, pp. 85, 274.
4. Humphrey, Watts S., "Year 2000 Readiness Checklists," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, July 1998, pp. 24-28.

Coming Events

Sixth Annual ISO 9000 Conference

Dates: March 22-23, 1999
Location: Atlanta, Ga.
Theme: "New Direction for Continuous Improvement"
Sponsor: American Society for Quality
Topics: Upcoming Revisions to the International Organization for Standardization (ISO) 9000 Quality Management Standards, Technological Shortcuts to Accurate Process Documentation, Case Studies of Corrective Action Resulting in New Levels of Effectiveness, Cost Benefits of ISO 9000 to Secure Management Buy-In, and How Even Small Companies Can Use Internal Auditors Effectively.
To receive full conference program, call: 800-248-1946, ask for item B0424. Mention priority code CEJAPR8 to speed order.
Internet: http://www.asq.org/products/conf.html

Call for Papers: The Sixth International Symposium on Software Metrics

Dates: Nov. 5-6, 1999
Location: Boca Raton, Fla.
Theme: "Taking the Measure of New Technology"
Topics: The theme of the conference will focus on the application of measurement (through empirical studies) to understand and manage new software technologies (including their related tools and processes), such as commercial-off-the-shelf-based development, software architectures, object-oriented development, and Web-based applications.
Abstracts due: April 1, 1999
Contact: David Card, General Chairman, Software Productivity Consortium, 115 Windward Way, Indian Harbour, FL 32937
Voice: 703-742-7199
Fax: 703-742-7200

Call for Papers: Thirteenth Annual Advanced Software Engineering Education and Training (ASEET) Symposium

Dates: July 26-29, 1999
Location: U.S. Air Force Academy, Colorado Springs, Colo.
Theme: "Ada in the 21st Century: Academic, Government, and Industry Perspectives"
Topics: For a complete list of topics, refer to the Web page listed below.
Abstracts due: April 30, 1999
Full paper due: June 25, 1999
Send all submissions (MS Word or Adobe PDF) via E-mail to Lt. Col. J.A. Hamilton Jr. or Dr. Martin C. Carlisle.
E-mail: hamiltoj@spawar.navy.mil
E-mail: mcc@cs.usafa.af.mil
Internet: http://www.acm.org/sigada/aseet

INCOSE '99

Dates: June 6-10, 1999
Location: The Stakis Brighton Metropole Hotel, Brighton, England
Sponsor: The United Kingdom Chapter of the International Council on Systems Engineering (INCOSE).
Theme: "Systems Engineering: Sharing the Future"
Topics: INCOSE '99 seeks to exploit the confluence and synergy that we are seeing between various key issues being addressed by INCOSE worldwide. Come and share in the challenges of the breadth of applications, the diversity of techniques, and the overlap that systems engineering has with other disciplines.
Contact: Cass Jones, Professional Conference Management, Inc., 7916 Convoy Court, San Diego, CA 92111
Voice: 619-565-9921
Fax: 619-565-9954
E-mail: pcminc@pcmisandiego.com
Internet: http://www.incose.org.uk

5th Annual Joint Aerospace Weapon Systems Support, Sensors, and Simulation Symposium and Exhibition

Dates: June 13-17, 1999
Location: San Diego, Calif.
Theme: "Making Information Work for the Warfighter"
Sponsors: Embedded Computer Resources Support Improvement Program, U.S. Air Force, U.S. Army, U.S. Navy, and U.S. Marine Corps.
Contact: Dana Dovenbarger
Voice: 801-777-7411 DSN 777-7741 or 801-698-0132
Fax: 801-775-4932
E-mail: dovenbad@software.hill.af.mil
Internet: http://www.jawswg.hill.af.mil

Configuration Management
Coming of Age in the Year 2000

Clive Burrows
Ovum, Ltd.

Some estimates show that the market for configuration management (CM) tools and services now exceeds $1 billion per year and is still growing rapidly. The market size and growth has led to many of the founding companies in the CM market being acquired by larger companies with no history of involvement in CM. This article summarizes the proven capabilities of CM tools that have created a market of this size and reviews the potential areas for future development—most notably CM control of Web site content. It also summarizes the potential impact of these acquisitions on users and notes the rise of new companies entering the market with good cost-effective products.

Configuration management is the key to managing and controlling the highly complex software projects being developed today. CM tools have developed from simple version-control systems targeted at individual developers into systems capable of managing developments by large teams operating at multiple sites around the world. The variety of tools offered means that you can be sure to find one that is a close match to your individual needs.

The need for this degree of team support has grown in response to time pressures on software development and the increasing need to manage multiple changes to the same software at the same time. For example, resolving year 2000 (Y2K) compliance problems while still developing new features and fixing bugs.

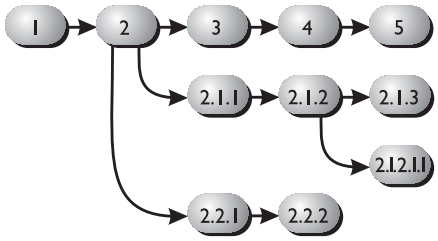
As a result, the market for CM tools exceeded $1 billion in 1998, and many of the smaller companies that created the modern CM capability have been swallowed up by the large players in the software arena. Even so, the growth in the CM market is so strong that there is still room for new entrants to introduce new products and to compete successfully.

Key Feature of CM Tools

The key capabilities of CM tools are the identification and control of software and software-related components as they change over time.

For most users, the main issue is the tool's ability to support a project team that develops software in a single repository, even though individual members of the team may be at different locations connected by a network. Individual members of the team need to be able to undertake the tasks assigned to them without interference from other team members. However, as each task is completed, the results need to be made available to other team members to assimilate into their own work at a time of their choosing.

Figure 1. *A typical revision sequence.*



Version control, the original CM requirement, maintains a history of the changes to a component as it evolves over time and allows users access to a particular version—not just the last version created.

Parallel Working

Originally, when team sizes were small, the accepted wisdom required CM systems to prevent several users from attempting to change the same component at the same time. Many organizations still attempt to operate with this "safe" development discipline. However, two important developments have combined to allow multiple users to change the same root component without undue risk.

The first development was the extension of version-numbering systems to support branches and therefore parallel developments by different users along different branches of the version tree. Figure 1 shows a typical revision sequence and numbering scheme including the creation of new development branches.

The second development has been the availability of merge tools with a strong graphical interface, which assist the resolution of conflicts when changes made on different branches are merged back into the main development stream. The best merge tools can relate all changes to a common ancestor (in Figure 1, revision 2 is the common ancestor of all branches emanating from the original single stream of development) and can therefore accept many changes automatically with only conflicting changes being raised for user attention.

The capability of modern merge tools is now so strong that users can become tempted to accept the tools' automatic resolutions and omit essential testing processes. This is not recommended.

The branching approach to parallel development is used in a few tools to support the cloning of software repositories across a number of geographically distributed sites or even companies, e.g., for consortium projects. In these circumstances, the issue becomes the level of capability to support the periodic synchronization of databases between sites.

An earlier version of this article was published in SIGS's Component Strategies, *July 1998.*

The individual cloned repositories are never at an identical state, but over time all changes are applied to all sites. The repositories may be different at any time, but the differences never become too great and are always controlled at a manageable level.

Web Interface

New developments in almost all CM tools are the provision of some CM functionality through a Web browser interface. There is still a big difference between the CM tools in the degree of support provided. Apart from those with no Web access capability, the minimum support tends to be for problem and change management systems and for information reporting systems, i.e., all those aspects with a minimum requirement for data transfer. Only a few tools provide a check-in, check-out facility to access and modify files through a Web interface.

Some tools prefer the higher bandwidth solution of a "cut down" client for home CM access via a modem into the organization's network.

Change Management

Change management features address the issues of problem tracking and change control and the presentation and analysis of management information derived from these sources. Gathering management information is greatly simplified if change features are part of the CM system—without them, complex cross-references between different databases are required, and full navigation and searching may not be possible.

Unfortunately, many CM vendors have developed their own add-on capability in this area using new development tools, different databases, and even a different style of user interface. In some cases, the only area of commonality is the product "badge" name created by the marketing department.

Build and Release Support

Building systems can take days, and an inefficient build process can waste hours of developer time, particularly during testing and integration when you may need to build the whole system to test a small change. An intelligent build process can reduce build times dramatically by reusing partially built items from previous builds.

Release support allows developers to track which users have which versions of which components and, therefore, to be sure which of those will be affected by a particular change.

Process Management

Many users, particularly those seeking an external quality approval such as ISO 9000 or a particular Software Engineering Institute Capability Maturity Model level, have standard development processes they expect their development teams to follow. In the past, this has often involved considerable bureaucratic paperwork procedures, which are generally resented and ignored by developers.

The process management features in CM tools allow the developer to ensure that components progress through chosen lifecycle phases before being released. An example of this is to help ensure that testing and quality assurance occur before release. The tools take a wide range of approaches to process management, and it is important to select one that suits the culture of your organization or the new culture you wish to introduce.

New Developments

Year 2000

This is an immediate, if short-term, opportunity for CM tool vendors. Traditional CM support for multiple streams of development activity is becoming increasingly important as the millennium approaches. Y2K conformance issues must be addressed in parallel with new developments and with bug fixes. In addition, governmental and regulatory action to open up markets is requiring European utilities—and in the near future, U.S. utilities—to modify their systems to support competition. Financial institutions and multinational corporations worldwide need systems to cater to Euro currency. This degree of parallel working is not sustainable without strong CM support.

Y2K activities also have a different characteristic from normal development work. About 60 percent of all Y2K effort is spent on testing. As problems are found, traditional change management processes are applied to ensure that all changes are pursued to completion and that adequate retesting takes place. Individual modules need to have their compliance status logged to allow a full audit trail to be established. If a new Y2K issue is identified, selective retesting of previously "compliant" modules may be necessary. CM tools have features to support all these requirements.

Although Y2K issues are seen by most users as short term, the emphasis on test management will provide long-term benefits for all users of CM tools well beyond Dec. 31, 1999.

Web Management

CM support for Web and particularly Intranet pages and their embedded objects is creating an important new market for the vendors of CM tools, which in time *could* exceed the size of the market for managing software development.

To develop this new market opportunity, vendors have to address three main issues:

- Different characteristics associated with Intranet and Web management.
- Different user skills and profile of Intranet and Web developers.
- New marketing issues to be addressed to attack this market.

The different characteristics include sheer size—the number of pages being managed for an Intranet site can be more than 100,000, implying a requirement to manage more than 500,000 objects. By contrast, 10,000 software modules would represent an extremely large software development program.

The rate of change for Intranet pages is high, but the lifetime of a typical page can be relatively short. This is associated with an extremely high level of build and release operations. It is not unusual for a large Intranet site to be changed on a daily basis, whereas for large software systems, a new release every month would be regarded as an indicator of serious instability.

Intranet sites can contain commercially and legally sensitive information. For protection against lawsuits, not only

does a full audit trail of approvals need to be kept, it also is essential that a precise copy of the Intranet site on a particular date can be faithfully re-created when required. The volume of configuration information needed to accomplish this is phenomenal. In comparison to that needed for software configuration management, it is like comparing the number of Internet servers in the universe to the number of servers in a single company.

All CM vendors can truly claim a capability to address the individual issues raised here—this is why they believe that, in time, CM Intranet management will be a bigger gold mine than CM for software development. The issues, however, are not all related to functionality. There also are scalability issues. Vendors that attack the Intranet market will need a strong partnership with pioneering users to demonstrate this scalability. Other vendors can still market their smaller-scale systems to manage the smaller public Web sites.

A large information technology (IT) development might involve hundreds of people, nearly all assigned full time to the project. A large Intranet site might have more than 10,000 contributors, each probably spending no more than a few hours each month updating elements of the Intranet pages to which they contribute. A much smaller group of webmasters (broadly equivalent to build managers in an IT project) are responsible for managing and publishing the Web and Intranet content.

None of these people are CM aware, nor do they want to be. The providers of Web content are most likely to use Notepad or a word processing package, whereas the webmasters will certainly use Web-based tools for their day-to-day tasks. It is essential in this environment that vendors integrate their CM functionality seamlessly within the tools used for day-to-day tasks. It also is highly desirable that CM terminology be minimized, for example, by referring to a third draft rather than to Version 3.

Access to the CM functionality needed by webmasters should be via a Web interface to minimize the need to switch styles of working.

Most CM tools already have a degree of support for CM functionality via the Web and can readily integrate with a wide range of third-party tools. Some vendors have gone so far as to repackage their software CM systems (including the product documentation) with additional functionality to create a product targeted directly at this market (Web Integrity from MKS, WebSynergy from Continuus, and StarTeam from StarBase are examples). Other vendors are doing little more than make reference to Web issues in their promotional literature.

To fully exploit the Web management market, CM vendors have three other hurdles to jump—not technical issues but marketing issues.

First, they do not know how to find the potential buyers. In a software environment, vendors often find potential buyers coming to them, and they gain good links with IT managers and other project managers as the word spreads that these products really work. The potential buyers for Web management CM tools are not so easily found.

The reasons for this are the second and third hurdles: The potential buyers are not aware that they have a problem, and even if they were aware, they have no inkling that CM is the answer.

It has taken six years for CM vendors to establish a software market for capabilities beyond simple version control. Even so, their market penetration in an area they know well is still only 20 percent. CM vendors have a capability to satisfy a latent market need for managing Web and particularly Intranet sites, but they first have to establish market awareness. Currently, the potential Web market could dwarf their traditional software market, but for now they have 0.01 percent of a lot—which does not amount to much.

Documentation Support

The competition CM vendors face in the Web management market will be from document management systems. Although these systems have zero version control capability, they are perceived as being closer to the needs of Web site managers. And this perception is true. Document management systems *are* closer to the needs of Web site managers than CM systems—but only in the sense of their position in a queue of issues that have so far failed to get the attention of Web site managers.

Document management systems do have features that assist webmasters but these are not in conflict with the features offered by CM tools. In fact, document management systems are extremely poor at version management and related CM issues and would benefit from closer links with CM tools.

Many CM tools are starting to offer support for documentation development via integration with such products as Framemaker. The version and configuration support for documentation tends to be at a relatively high level (chapters or major document sections), but it also includes support for a document "build" process. In the future, this support will extend to CM management of embedded objects within the document, e.g., diagrams and pictures. This management of embedded objects in documents is closely allied to similar issues within Web pages. The vendors' development money is going into solving the Web management issues, but the spin-off will be a much stronger capability to manage many types of complex documents.

Tool Integration

Historically, most CM systems have targeted the management of software sources held in ASCII files. The scope of support provided for this environment is not necessarily available to users of Integrated Development Environments (IDEs) or fourth-generation languages (4GLs), which are not file based but repository based. There is little that CM vendors can do to add full CM value to products in this group until they are given access to the elements within the repository. This is starting to happen as users of these environments start to suffer the problems first encountered by COBOL and C developers. The IDE and 4GL systems that fail to offer links with CM systems are adding to the development risks of their customers instead of reducing them as they promise.

Project Management

A new trend, so far supported by just a few products, is to use the development progress information held within the CM system to link with project management systems such as Microsoft Project. In principle, this should add an extra dimension to the progress information available to project managers. This is not yet the case, but the process has started. And in the future, the scope of what is considered to be CM will undoubtedly include strong links with project management systems.

The Players

The original players in this market were small, innovative companies that jointly created a $1 billion market from a small base of users who were previously only familiar with free version-control software.

As user demands for team support grew, the inadequacies of the free version-control software became apparent (free software rarely provides value for money). The new products offered much more capability, and the companies developing them were strongly focused on CM alone. By 1995, these companies were well established, had a strong user base, and were doing business in a market that was still growing strongly. Some became quoted in the NASDAQ℠ (National Association of Securities Dealers Automated Quotations) index, while others preferred to remain private.

Neither choice changed the outcome. Almost all the founding companies of CM are now owned by "software conglomerates."

- TeamOne was bought by Legent, and their TeamNet product was renamed Endevor/WSX. Legent, in turn, was bought by Computer Associates, and Endevor/WSX was renamed Endevor/Unix.
- Platinum bought Softool, and for a change, did not rebrand the CCC product range.
- Atria was bought by Pure Software, and subsequently, Pure Atria was bought by Rational. Products such as DDTS were badged with the Pure name (PureDDTS) and subsequently with the Clear name (ClearDDTS) to establish an association, however loose, with the ClearCase CM product.
- Intersolv bought SQL Software and rebranded PCMS Dimensions as PVCS Process Manager and then confused the buyers when it sold Process Manager only with a package of other PVCS products and named the package PVCS Dimensions.

New JTA Version Announced

The Architecture Coordination Council (ACC) met May 28, 1998 and approved the Technical Architecture Steering Group Majority Position Document. The implementation memorandum dated Nov. 30, 1998 signed by the trichairs of the ACC makes Joint Technical Architecture (JTA), Version 2.0 effective for use immediately, superseding JTA, Version 1.0.

The JTA is a document that mandates the minimum set of standards and guidelines for the acquisition of all Department of Defense (DoD) systems that produce, use, or exchange information. The JTA shall be used by anyone involved in the management, development, or acquisition of new or improved systems within DoD.

The memorandum, JTA, Version 2.0, and related information is available on the Data and Analysis Center for Software Web site at http://www.dacs.dtic.mil/databases/url/key.hts?keycode=2024.

The JTA provides DoD with the basis for seamless interoperability of information technology systems. The JTA defines the service areas, interfaces, and standards (JTA elements) applicable to all DoD systems, and its adoption is mandated for the management, development, and acquisition of new or improved systems throughout DoD.

The JTA consists of two main parts: the JTA core and the JTA annexes. The JTA core contains the minimum set of JTA elements applicable to all DoD systems to support interoperability. The JTA annexes contain additional JTA elements applicable to specific functional domains (families of systems).

The JTA is a living document and will continue to evolve with the technologies, marketplace, and associated standards upon which it is based.

Within just a few months, Intersolv was acquired by Micro Focus.

- In December 1998, a relatively new entry to the market, Tower Concepts (Razor), was acquired by another privately owned company, Visible Systems Corporation.

Most acquisitions have been by companies with little experience in CM that aim to buy a stake in this market. The acquisition cycle is not yet complete.

While acquisitions of this nature can introduce additional funding for product development and synergy with related products, the end result is not always good news for the user.

- After the acquisition, there is usually a period of quiescence while the buying company tries to understand what it has bought and the bought company tries to understand its new environment.
- After the quiescent period, expenditure on marketing and related issues tend to get immediate priority over technical development issues—the new owners want a return on their investment quickly.
- Support is rationalized, i.e., reduced, by integration with established "help" desks, which lengthens lines of communication between the user and the people who know what they are talking about.
- New development expenditure becomes directed at integrations with the conglomerate's "Enterprise Support" products. They do this in the name of providing wider support for all users, but in reality, it benefits the conglomerate's existing customers by giving them a CM capability. The CM user gets offered related "enterprise" products that they do not want.
- Overseas dealers are often disenfranchised in favor of the conglomerate's local office, which gives established users even less support.
- Small users are no longer nurtured as the big corporate sell takes over—10 licenses no longer motivate the salesman who now needs 50 license deals to keep on target.

Of course, the new owners are still making money—the increase in sales outlets and sales resource makes this almost inevitable—but they do not have it all their way. One effect of these ownership changes and a sure sign of a growing market has been the emergence of a new group of small companies (Perforce, StarBase, and Tower Concepts—still small despite being acquired by Visible Systems Corporation, another privately owned company) that targets just those project groups favored by the developers of this market with proven developer-oriented messages.

Conclusion

There are over 50 companies that offer products to meet CM needs. Most are expanding their business and profits, and there is no sign of this declining. The main competition for CM vendors is still the users' lack of awareness of the success and capability of this technology. People do not wake up in the morning with "It's time to buy a CM tool" at the top of their to-do list. Instead, they wait for a foreseeable and inevitable disaster to kick-start the process—and then they usually buy from the first company they call. Make sure you do not do this by being an educated consumer. ◆

About the Author

Clive Burrows is principal evaluator of configuration management products for Ovum in London, England. He is the author of four Ovum reports on this subject, the most recent being *Ovum Evaluates: Configuration Management* (June 1998).

E-mail: clive_burrows@compuserve.com.

Applying Management Reserve to Software Project Management

Walter H. Lipke
Oklahoma City Air Logistics Center, Directorate of Aircraft Maintenance, Software Division

Today's standard of practice for managing a project's management reserve is an art form. In an effort to make this activity more scientific, the Software Division at the Oklahoma City Air Logistics Center has begun to use an extension of the Cost/Schedule Control Systems Criteria (C/SCSC) [1] technique to manage the reserve components of a software project to achieve the expected completion date and cost.

Since 1985, the Test Program Set (TPS) development activities within the Software Division at the Oklahoma City Air Logistics Center have been performing project management using C/SCSC methods. Initially, the application of C/SCSC management techniques was not thought to be suitable for software. In general, in 1985, only weapons systems program offices involved with major acquisitions employed C/SCSC management. For anything less than a major acquisition, its use was considered to be overly burdensome. However, it was the Software Division's belief that this management system provided advantages over the use of Gantt (milestone) charts that were typical for software projects in 1985. Even today, these charts are extensively used, although the use of earned value is gaining some popularity. The failing with Gantt charts is that managers have no way to connect the outlay of money to the project plan and to the project production; therefore, software managers who use Gantt charts do not have a good understanding of their project's status.

Our initial application of C/SCSC management was crude at best; however, with the performance of several TPS development projects, including the B-1 and B-2 aircraft weapons systems, the methods have evolved and improved and become increasingly more sophisticated. The work breakdown structure (WBS) presently employed bears little resemblance to the one first used in 1985. The earned-value system used today is an order of magnitude more resolute than the system first used in our employment of C/SCSC. Initially, we used only four earned-value elements, regardless of the project requirements; today, we have as few as 10 and as many as 64.

The C/SCSC methods of project management have served the Software Division well. The method is applied at the individual TPS developer level and is aggregated by team lead, total project, and higher organizational levels for various management and customer status reports. Employed in this manner, the method is quite flexible and becomes an extremely powerful management tool. In the 13 years C/SCSC techniques have been used, we have not experienced a single overall TPS development project slippage or cost overrun.

Until a few years ago, our application of C/SCSC project management did not segregate management reserve (MR) into quantifiable management elements. Although the project plan accounted for the risk in meeting cost and schedule, the MR was integrated into the earned-value system, and thus, its management became unrecognizable. Figure 1 illustrates this point. It shows that the budgeted cost of work scheduled (BCWS) line increases with time until the project completes as planned, indicated by budget at completion (BAC). The difference between BAC and the total project cost and negotiated completion date are the project's MR. As previously explained, our initial application of the C/SCSC method equated BAC to the total project cost and negotiated completion and thereby eliminated the possibility of managing the reserve.

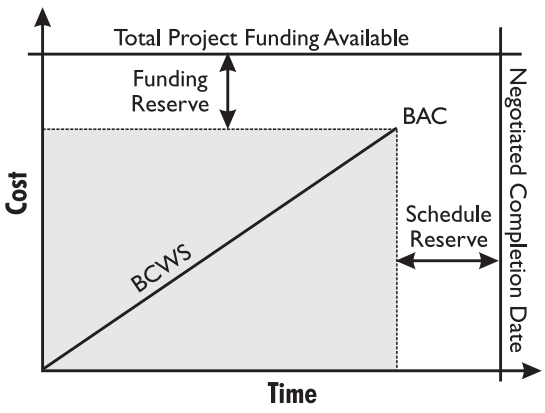
C/SCSC Refresher

A review of C/SCSC terminology and computations will be required to better understand the remainder of this article. The fundamental elements follow. For additional information concerning these formulas and terms, refer to [1].

- BCWS – budgeted cost of work scheduled.
- BCWP – budgeted cost of work performed.
- ACWP – actual cost of work performed.
- BAC – budget at completion.

EAC (estimate at completion) = ACWP (cumulative) + CPI^{-1} [BAC – BCWP (cumulative)]

Figure 1. *Management reserve.*



CPI (cost performance index) = BCWP / ACWP
(Greater than 1 is good.)

TCPI (to complete performance index) = [BAC – BCWP (cumulative)] / [EAC – ACWP (cumulative)]
(Greater than 1 is good.)

SPI (schedule performance index) = BCWP / BCWS
(Greater than 1 is good.)

TCSI (to complete schedule index) = [BAC – BCWS (cumulative)] / [BAC – BCWP (cumulative)] *(Greater than 1 is good.)*

In review, C/SCSC evaluates the calculations of schedule in units of dollars, i.e., cost, rather than in units of time. Figure 2 is an example of a project that is executing behind schedule. Note the Now vertical line. For this example, C/SCSC measures, in units of dollars, the amount project performance lags behind schedule by schedule variance (BCWP – BCWS). Extrapolation of the ACWP line to the calculated EAC value graphically projects cost overrun, e.g., the difference between EAC and BAC. Also, graphical extrapolation of the BCWP line to the BAC value projects schedule slippage in units of time.

Management Reserve Indicators

Some of the desirable yet difficult to develop characteristics considered in the development of the MR indicators and analysis tools were

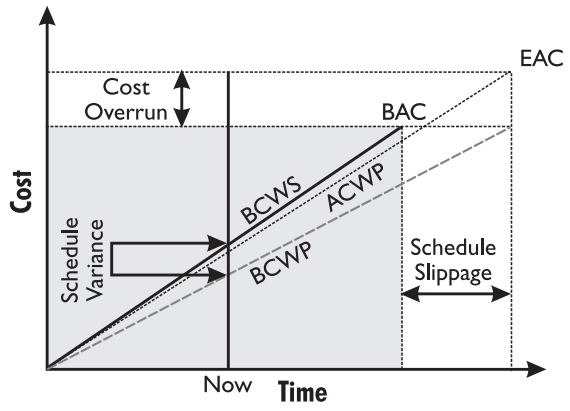
- Similar appearance for each component.
- Simple visual analysis—readily understood "stoplight" (red, yellow, or green) conditions.
- Simple or no project tailoring required.
- Simple calculations.
- Usefulness in project management.

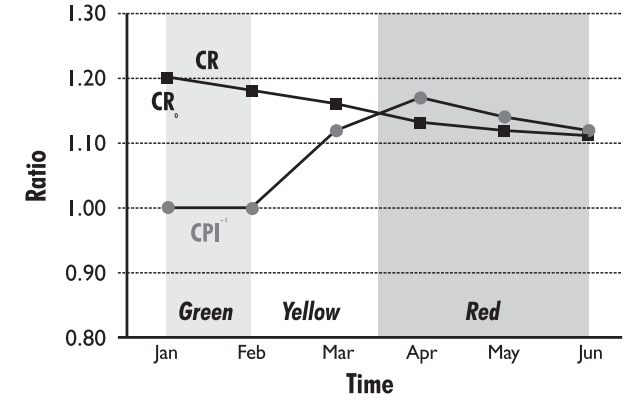
We believe that the indicators and the analysis tools we developed satisfy the above characteristics. The indicators are

- Cost ratio vs. CPI^{-1}.
- Schedule ratio vs. SPI^{-1}.

Cost ratio is total funding available (TFA) for the project divided by BAC, where TFA is the sum of BAC and funding reserve. Schedule ratio is negotiated period of performance (NPOP) divided by planned period of performance (PPOP); the difference of NPOP and PPOP is schedule reserve. For clarification, the ratio formulas are

Figure 2. *Cost and schedule analysis.*





Green: Project can be completed within planned cost and may have CR remaining. ($CPI^{-1} \leq 1.0$ and $\leq CR$)
Yellow: Project can be completed within funding available. ($CPI^{-1} > 1.0$ and $\leq CR$)
Red: Project cannot be completed within funding if present conditions continue. ($CPI^{-1} > CR$)

(Note: CR varies from CR_0 due to CR used to perform non-WBS activity.)

Figure 3. *Cost ratio vs. CPI^{-1}.*

Cost Ratio = TFA / BAC (dollars)

Schedule Ratio = NPOP / PPOP (time)

Both the cost and the schedule indices (CPI and SPI) provide information about the cumulative performance of the project at a specific point in time. Also, both indices similarly indicate good performance by a number equal to or greater than one. It was observed that the inverse of the indices could be compared to the corresponding ratios of negotiated vs. planned values for cost and schedule. If the reciprocal of the index value is greater than one, the project manager should be concerned because the project is consuming MR. The level of the manager's concern can be determined by comparing the index value to the appropriate ratio. If the index value reciprocal exceeds its corresponding ratio, the manager knows the project cannot meet the customer's expectations without corrective measures.

These indicators are graphically portrayed as a time trend (Figures 3 and 4). Conceptually, the graphs of the two indicators are identical. If the project is performing such that CPI^{-1} and SPI^{-1} are less than their respective cost and schedule ratios, the project is in good shape. If this situation continues, the project will complete on time and within the allocated cost. If both CPI^{-1} and SPI^{-1} remain at the value of 1.0, the project is expected to complete as planned—a project perfectly executed.

The differences between the representation of the two indicators are small. The only significant difference is that cost ratio has the possibility of varying, and thus, its initial value is denoted as "CR" with an "o" subscript. The reason cost ratio varies and schedule ratio does not is because of the way C/SCSC accounts for cost and schedule. The use of

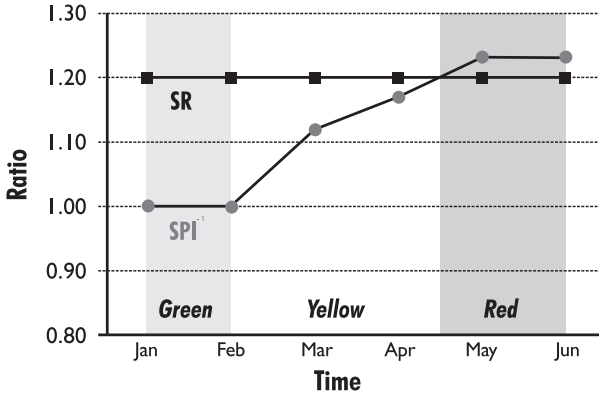
schedule reserve is accounted for by expending effort that does not gain earned value, i.e., BCWS marches on with time, but BCWP only does so by increasing earned value. However, cost is a different matter; the use of funding reserve is not reflected in ACWP, BCWS, or BCWP.

A nightmare for software project managers is "extras" thrown at them by the customer. Of course, revised requirements are supposed to be renegotiated and reflected by a revised project baseline that includes a new completion date and changed cost. However, many times the "requirements creep" seems so trivial that project managers forego the perfect practice and merely adjust their funding reserve to account for the change. For many situations, the effort required to re-baseline the project and negotiate the change is far greater than the amount of reserve lost. As an internal practice, we advise customers that changes are being accrued and that we reserve the right to negotiate them once it is apparent the effort to do so is worthwhile; however, until payment occurs for revised requirements, the reduction in funding reserve will be reflected in decreased TFA and thus a lower cost ratio.

Other than the variability of the cost ratio, the graphical appearance and analysis are virtually identical. The conditions to determine the health of the project are simple and easy to recognize. If CPI^{-1} and SPI^{-1} are equal to or less than 1.0, the project can be completed as planned, and the stoplight indicators would be green. And, if the cost of the effort expended for unplanned requirements does not totally consume the funding reserve, some funding is expected to remain at project completion (a project manager's delight—cash bonuses for everyone on the project). If CPI^{-1} and SPI^{-1} are computed to be between the value of 1.0 and their respective ratios, the stoplight indicator is yellow—the project is not performing as well as anticipated but is still executable (project manager and employees get to keep their jobs). The last condition, i.e., the red indicator, is evident when CPI^{-1} and SPI^{-1} exceed their respective reserve ratios. The project cannot be completed in the red dimension if the present conditions continue—the negotiated cost or schedule is expected to be exceeded (a bad situation for those involved).

Table 1. *Recovery strategies.*

CR vs. CPI^{-1}	SR vs. SPI^{-1}	Action
Green	Green	Reward employees.
Green	Yellow	Increase OT.
Green	Red	Increase OT or people.
Yellow	Green	Decrease OT.
Yellow	Yellow	Review and adjust assignments.
Yellow	Red	Adjust assignments; consider negotiation (schedule).
Red	Green	Decrease OT or people.
Red	Yellow	Adjust assignments; consider negotiation (funding).
Red	Red	Negotiation (funding, schedule, requirements); fire manager.



Green: Project can be completed within planned time. ($SPI^{-1} \leq 1.0$)
Yellow: Project can be completed within negotiated time. ($SPI^{-1} > 1.0$ and $\leq SR$)
Red: Project cannot be completed within negotiated time if present conditions continue. ($SPI^{-1} > SR$)

(Note: SR does not vary. Use of SR for non-WBS activity is accounted for in SPI, i.e., BCWP.)

Figure 4. *Schedule ratio vs. SPI^{-1}.*

Management Use

The next step is to determine the appropriate management action when conditions are other than green. Managers have a choice of four possible strategies to recover a project:

- Adjust overtime or number of employees.
- Realign employees to increase efficiency.
- Reduce performance requirements.
- Negotiate additional funding or schedule.

Generally, strategies 1 and 2 are within the project manager's prerogative and are much preferred, whereas strategies 3 and 4 require unpleasant negotiation with the customer. Application of strategies 3 and 4 are to be used as a last resort because they build a negative image that impacts future business with that customer and others who might contact your customer as a reference.

Table 1 aggregates all the combinations of conditions possible for the two indicators and associates each combination with a specific recommended management action. Certainly, for multiyear projects, if both indicators are consistently green, the manager should reward employees—the program has had good planning and good execution. It is worthy to note that if one of the indicators is green, the project is recoverable. If one of the indicators is yellow and the other is red, negotiation must be considered (recovery miracles do not often happen). If both indicators are red and the project is far enough along for everyone to know red means failure is imminent, there is no alternative—cost and schedule must be renegotiated, or the project must be willing to absorb the financial loss and endure the humiliation of a major schedule slippage. Under these conditions, managers and employees are at risk to be replaced.

A viable action under project manager control is to realign employees to increase their efficiency; however, realignment requires in-depth understanding of the strengths and weak-

nesses of the staff and the roles the project requires. To incorrectly match staff to new roles can seriously impact performance efficiency. The seriousness of the staff deficiencies and the length of time remaining on the project are to be considered in taking employee realignment actions.

Calculations

Two of the more manageable strategies under the sole control of the project manager are varying overtime and number of employees, for which a few helpful formulas are given in the following section. The equations are presented first for schedule recovery, then cost recovery. Bear in mind that reserve funding is used for schedule recovery; people and overtime are increased. For cost recovery, the opposite must occur; people and overtime are decreased at the expense of schedule reserve. It also is important to remember that the formulas are constructed to resolve the predicted schedule or cost overrun by adjusting either staffing or overtime, not both. In other words, the results of the computations can be used to establish the bounds for the management action.

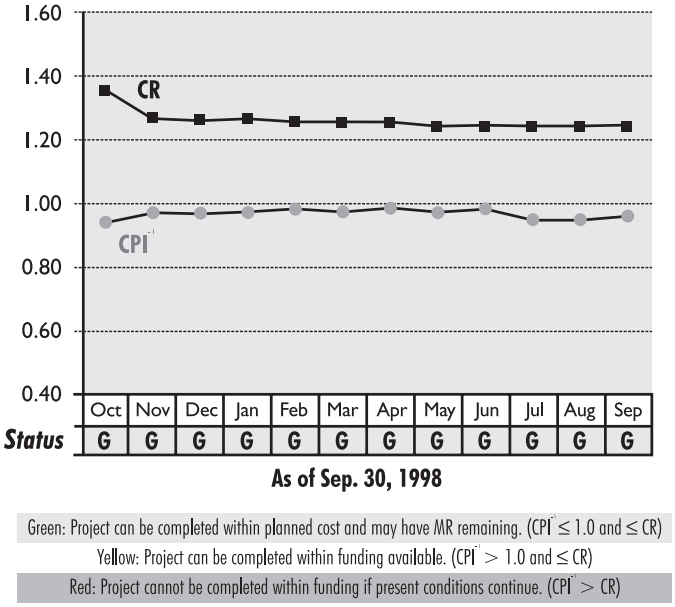
For Schedule Recovery

To determine the average number of employees needed for the remainder of the project, calculate

E_{SR} = [BAC – BCWP (cumulative)] / [CAR • project time remaining (years)], where CAR (Cost Accrual Rate) = total average cost per person / year. *(The term [BAC – BCWP (cumulative)] represents the project's remaining schedule in dollars.)*

The number computed should be larger than the initial average staffing number. The difference in the two numbers provides information regarding the adjustments needed in the project's man-loading profile. If TFA is used instead of BAC, the project can be expected to use all the funding reserve.

Figure 5. *Prototype project: cost ratio vs. CPI^{-1}.*



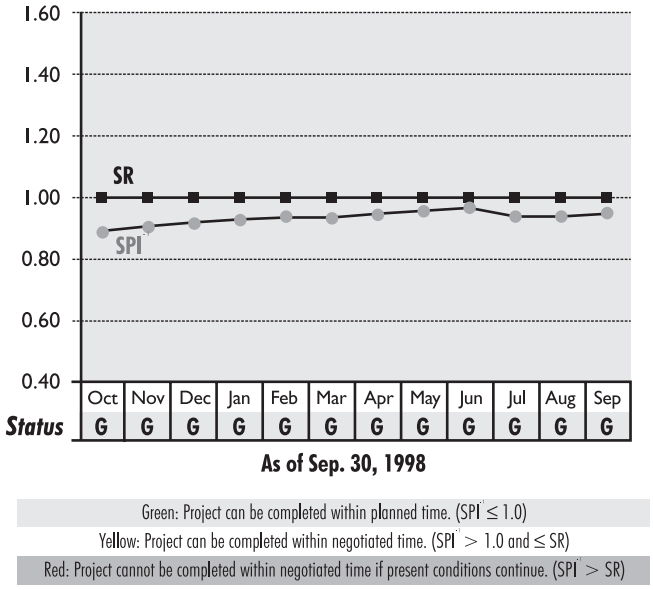


Figure 6. *Prototype project: schedule ratio vs. SPI^{-1}.*

To determine the overtime (OT) needed, calculate $TCSI^{-1}$ to determine the ratio of the actual to the planned schedule remaining—the ratio will be larger than one when the project is behind schedule. The computed value of $TCSI^{-1}$ is then used in the calculation of the OT rate required for the remainder of the project. The elevated OT rate is computed using the following equation.

$OT_{SR} = (TCSI^{-1}) \bullet (1+OT_p) - 1$, where OT_p is the planned OT rate.

The expectation is that by working at this rate, employees will complete the project on the planned date. If the OT rate exceeds what is considered a "burn out" threshold, an increase in staffing should be considered. If TFA is substituted for BAC, the OT rate required will be less; however, all schedule reserve is expected to be used.

For Cost Recovery

To determine the average number of employees needed for the remainder of the project, calculate

E_{CR} = [BAC – ACWP (cumulative)] / [CAR • Project time remaining (years)], where CAR is the same as for the schedule calculation. *(The term [BAC – ACWP (cumulative)] represents the remaining project funds.)*

The number computed should be smaller than the initial average staffing number. Similar to schedule recovery, the difference in the two numbers provides useful information concerning the man-loading change needed. If TFA is substituted for BAC, the project can be expected to use all the funding reserve.

The overtime calculation for cost recovery is very similar to the calculation presented previously for schedule recovery. First, calculate $TCPI^{-1}$ to determine the ratio of the actual to the planned funding remaining—the ratio will be smaller

than one when earned-value efficiency is poor. Analogous to schedule recovery, the value of $TCPI^{-1}$ is used in the computation of the OT rate for cost recovery required for the remainder of the project. The adjusted OT rate is calculated using the following equation.

$$OT_{CR} = (TCPI^{-1}) \bullet (1+OT_p) - 1$$

The expectation is that by working at this reduced OT rate, employees will complete the project at the planned cost. If the OT calculation produces a negative number, the project *must* reduce its staffing. If TFA is substituted for BAC in the calculation, a smaller decrease in OT rate will result so as not to exceed the available funding reserve.

Project Application

Over the last year, we have been prototyping these management tools and ideas in a large development project. As can be seen in Figures 5 and 6, not much information about the usefulness of the tools can be stated; the project has performed too well. To date, no cost or schedule recovery has been required. However, a few observations can be made. Before the tools were developed, the only reserve component considered in project planning was funding. Figure 6 illustrates this point; the prototyping project has a schedule ratio of 1, thereby indicating the absence of schedule reserve. Because they recognize the value and reduced risk of having two dimensions of MR, our managers now pay much more attention to the schedule component. The new projects are being planned with consideration for schedule reserve.

Other Thoughts

In considering the application of these tools, you should recognize that considerable discretion is required. If applied in too rote a manner, especially early in a project, there is risk of tampering, e.g., overcorrection. Generally speaking, if yellow and sometimes even red indications occur early in the project, it is wise to merely look into the problem and wait for the next review before taking action.

Summary

The concepts presented are extensions of C/SCSC and are targeted to the effective use of MR. The tools presented provide simple visual aids to assess project health, which, in turn, leads to suggested management actions. Calculation formulas are also provided to further refine the recommended management action. This set of management tools should be easily applied by anyone who uses C/SCSC for software project management.

The prototyping of the tools performed to date does not provide sufficient information to show their usefulness. Even so, because we believe that the indicators, prescribed management actions, and formulas are conceptually sound, we are proceeding with their application to other projects. By expanding the application of the MR management technique this year, we expect to broaden our perspective by gaining additional inputs from several managers. ◆

Reference

1. Fleming, Quentin W., *Cost/Schedule Control Systems Criteria, The Management Guide to C/SCSC*, Probus, Chicago, 1988.

About the Author

Walter H. Lipke is the deputy chief of the Software Division at the Oklahoma City Air Logistics Center. The division comprises approximately 600 employees, most of whom are electronics engineers. He has 30 years experience in the development, maintenance, and management of TPS. In 1993, under his direction, the TPS and industrial automation functions of the division became the first Air Force software activity to achieve Software Engineering Institute Capability Maturity Model CMM Level 2. Likewise, in 1996, these functions became the first software activity in federal service to achieve SEI CMM Level 4 distinction. Recently, under his direction, the TPS and IA software functions achieved ISO 9001 and TickIT registration. He is a professional engineer with a master's degree in physics.

OC-ALC/LAS
Suite 2S12
3660 C Avenue
Tinker AFB, OK 73145-9144
Voice: 405-736-3335
Fax: 405-736-3345
E-mail: wlipke@lasmx.tinker.af.mil

Logical Event Contingency Planning for Y2K

Robert L. Moore and Roberta H. Krupit
Coastal Research and Technology, Inc.

Traditional contingency planning methods do not work well for logical events, such as year 2000 (Y2K) problems. And although there are many different types of logical event contingency plans, those that work well for some situations may be of no use for others. This article discusses the unique aspects of logical event contingency plans and helps you plan appropriate strategies to deal with logical events.

Any information technology (IT) system needs its computing resources to operate correctly (system integrity) and should maintain the value of its information (data integrity). Unfortunately, adverse events may damage data integrity or system integrity or both.

Contingency planning for IT systems focuses on preserving, enabling recovery, or the graceful degradation of system or data integrity. Unfortunately, contingency mechanisms that work in one adverse situation may be hopelessly inadequate in another. Consequently, different contingency plan types exist for different "disasters." The two primary types are physical and logical event plans.

Traditional IT contingency plans address physical events such as flood, fire, earthquake, war, or loss of power. Many sources discuss traditional contingency planning, which often emphasizes replication and physical separation to guard against physical disasters. Such traditional planning does not consider logical events.

A logical event (LE) strikes all sites that have similar information configurations (software, data, and firmware)—no matter how widely separated. The impacted system also may corrupt IT that is down the information flow stream. LE contingency planning has two subcategories: plans for incompletely understood systems and plans for well-understood systems. Because the Y2K problem is an LE, discussion of LE contingency planning is appropriate.

LE contingency planning borrows from systems security principles, which include auditing, system modeling, input and output validation, and instrumentation. This article outlines the properties that a contingency plan strives to preserve, suggests techniques for further investigation in Y2K LE contingency planning, and explains LE contingency plans with respect to traditional plans.

Y2K planning is about risk management. Risk management involves

- Identifying risks (potential threats and vulnerabilities).
- Analyzing risks by evaluating, categorizing, and prioritizing them.
- Planning for risks.

Contingency planning reduces to

- Creating mechanisms to identify the events that trigger contingency actions.
- Defining what the contingency actions are (either automatic or manually executed actions).
- Preparing the responsible parties by documenting who they are and by training them to be ready.

Weak Contingency Planning

An occasional protest to LE contingency planning is to offer a weak (or no) contingency plan followed with, "What more can be done? I can't plan for every possibility!" For example, a weak plan might call for canceling vacations and putting support personnel on call. This is merely a beginning that, unfortunately, does not take advantage of all available information.

Procedures to deal with a disruption should address the event's most probable serious consequences. Scale the plan to fit the consequences. For instance, if you run out of paper when printing E-mail, you acquire more paper—you do not buy a paper mill. However, if you run the U.S. Mint and you frequently run out of paper for money, perhaps you *should* buy a paper mill (or mint more coins). Planning for every event is impossible and counterproductive. It is prudent to analyze the adverse events that could occur (for example, Y2K problems) and construct mechanisms to preserve, enable recovery, or gracefully degrade system or data integrity.

Contingency Plans and Integrity

An IT system should be reliable, correct, and accurate. These integrity principles divide into two categories: data and system integrity. Data and system integrity include accuracy, completeness, consistency, timeliness, authenticity, authorization, precision, compliance with laws, regulations, organization policies, and procedures, and evidence that all of the proceeding properties are fulfilled [4]. IT that loses system or data integrity is worse than useless—it may be misleading and even dangerous. Integrity principles are fundamental requirements to reliable IT operation.

Contingency plans state how to manage the planned degradation, preservation, or restoration of system and data integrity. Not all adverse events impact all integrity principles. To create a contingency plan for a particular system, one addresses that system's requirements with respect to a catastrophe by documenting how to handle the integrity of specific requirements.

Physical Event Contingency Plans

Traditional contingency planning, whether from a computer-age viewpoint or not, assumes that problems are physical in nature. Adverse events take

the form of flood, fire, earthquake, war, terrorism, riot, hurricane, tornado, sabotage, loss of electricity, equipment failure, flu epidemic, and so on.

The distinguishing feature of physical problems is distance. A gas station explosion might impact business operations at a restaurant half a block away but will not impact operations at another gas station 100 miles away. Physical "disasters" have less direct impact as distance from the disaster increases.

Traditional contingency plans use the localization of physical disasters by emphasizing IT duplication in physically protected or remote locations. There is more to contingency planning than creating backups, of course. Other parts of physical event contingency planning include educating staff in contingency procedures, ensuring adequate management and security controls for operation during an event, cost analysis of recovery options, and mechanisms to rapidly transfer control to alternate sites. In any case, the ultimate safeguard in a physical event contingency plan is a remote operations center (ROC) that faithfully duplicates the capabilities of the primary operations center [7].

Fortunately, disaster planning literature covers the creation of an ROC. Consequently, even though the design, creation, and operation of an ROC are not easy, they are sufficiently documented that we need not revisit them here except in contrast to LE plans.

Logical Event Contingency Plans

The logical and physical worlds are fundamentally different, as are logical and physical events. A logical event, such as a bug in command and control (C^2) software that shuts down pumping operations at a gas station based on a quarterly pump maintenance query, will not impact a neighboring restaurant but may impact every gas station with the same C^2 software. Indeed, LEs may corrupt otherwise operational systems down the information stream from the impacted system. A plan to address physical events is unlikely to help with an LE (and vice versa).

Adverse logical events are the combination of threats and vulnerabilities, a combination that is possible due to bugs at some level (requirements, design, or implementation). Logical events can follow failure to anticipate possible data forms, hazards that could attack a system (like vulnerability to a virus), unintended intercomponent interactions, or design assumptions that eventually destabilize the system (Y2K).

Computer security events are typically LEs. Computer security events provide clear examples of logical "disasters" (such as viruses and Internet worms). Methods to mitigate security events are similar to methods for logical events in general and Y2K events in particular.

IT security focuses on preserving a system and its information content against malicious attempts to make data and resources unavailable, unreliable, inaccurate, or inefficient. IT security begins by trying to avoid events and concludes by building mechanisms to deal with events should they happen anyway. System development tries to avoid LE problems but uses contingency plans to address expected or unexpected potential threats. An LE contingency plan focuses on preserving a system and its information content against events that make data and resources unavailable, unreliable, inaccurate, or inefficient.

The comparison of the Y2K LE with a virus is indeed appropriate. Y2K consequences, although not malicious, degrade IT much like viruses.

How an LE impacts IT depends on the "distance" between the various components. Coupling, cohesion, and similarity of function and form determine distance. As understanding of a system increases, logical contingency plans grow from generic catch-all to specific plans. This understanding provides a means to develop more detailed solutions, such as "checksums" and logging mechanisms on automated actions and rapid debugging measures [4, 6].

Calculating Inputs from Valid Outputs or Outputs from Valid Inputs

Creating "checksums" from inputs and outputs depends on determining what valid inputs and outputs could be:

1. Using documentation, test cases and results, maintainer and user system knowledge, and accumulated live inputs and outputs surmise a set of acceptable inputs and outputs. This is a blueprint for what will and will not be allowed as inputs and outputs. This method is essentially anecdotal. However, it is also extremely practical in its simplicity (no special techniques or tools needed) and in its efficiency (the data is readily available). Take care that knowledge of historical inputs and outputs does not unintentionally exclude legitimate future input or output variations.
2. Calculate the weakest precondition (wp) or strongest post-condition (sp) to derive a set of conditions that must hold true either before or after execution, for the routine to function correctly. wp(S,R) is "the set of all states such that execution of [routine] S begun in any one of them is guaranteed to terminate in a finite amount of time in a state satisfying [expected result] R" [3]. Similarly, sp(S,I) represents that if [input condition] I is true, execution of S results in sp(S,I) true if S terminates [2].

Both wp and sp are obscure and have minimal automated support. However, they can be calculated almost completely mechanically. wp and sp are easiest to derive when good software engineering practices have gone into a routine (for instance, cyclomatic complexity is low, nesting is low, the routine is not too large, and the code is structured). Calculating wp and sp also requires more mathematical ingenuity when dealing with loops. The most practical method to determine wp or sp is to perform the calculations in sections with good software engineering or non-loop structures and to heuristically estimate what should be true in the spirit of wp and sp in the harder sections (see sidebar "Calculating Weakest Precondition for a Simple C Routine" on next page).

Y2K LE Contingency Planning

Most Y2K contingency planning is LE contingency planning. Writing a Y2K contingency plan depends on how much is known about a system. Y2K triage determines how much is known and divides Y2K-impacted systems into two categories:

Category 1 – Critical and noncritical systems that because of fiscal, technical, or time constraints or mission-related decisions will not be worked on with respect to Y2K.

Category 2 – Critical systems that will be examined,[1] possibly fixed, and tested and for which adequate resources exist to accomplish these tasks.

Category 1 systems need plans that address Y2K consequences at a macroscopic level (*generic* plans). Conversely, contingency plans for Category 2 systems are based on information derived from analyzing and possibly repairing the systems (*specific* plans).

Specific LE Contingency Plans

Compared to systems in Category 1, much is known about Category 2 systems. Information on what a Category 2 system does, how it does it, which sections of the code deliver what functions, etc., is usually available. This information is the basis of all subsequent contingency planning. It helps answer both technical (how do I guard against event X?) and management (which functions are important to me?) questions. (Generic LE contingency planning techniques, which are discussed later, may also apply to a system eligible for a specific plan, but not vice versa.) Specific plans feature an array of techniques[2,3] including input and output validation, auditing, and code instrumentation [4, 6]. An LE contingency plan's goals include the following:

- Detect problems quickly.
- Determine a specific cause for the problem.
- Determine a repair or, if no feasible repair exists, reduce or prevent further impact from the problem.
- Recover information about damage from the problem so that anything lost can be restored.
- Demonstrate due diligence in anticipating, avoiding, and mitigating problems.

The following subsections outline ideas for specific plans.

Input and Output Validation

Recovery time and cost are cut when problems are noticed quickly. One way to notice problems is to automatically validate inputs and outputs [4]. An understanding of inputs, outputs, and their interrelationship requires insight into both the data form and function[4] (see sidebar below). To use input and output validation as a Y2K defense

- Determine valid outputs or inputs or both, usually at a subroutine level. (Determining outputs is often easiest.)
- Given known, valid outputs, calculate valid inputs (or conversely, valid outputs from inputs (see sidebar below,

Calculating Weakest Precondition for a Simple C Routine

The following example shows how to calculate wp(S,R). (wp is discussed in the sidebar "Calculating Inputs from Valid Outputs or Outputs from Valid Inputs" on previous page). In this example, S is the program to be executed. R is a statement containing as much information as possible about what is hoped will be true after S executes. In this example, wp(S,R) is used to detect conditions that would cause R to not be true after S executes–that is, error conditions. Once wp(S,R) is calculated, it can be used as a built-in data check to see if errors will occur. Any data inputs that falsify wp(S,R) will cause the routine represented by "S" to produce unexpected results.

One way to use wp(S,R) is to place S in the context of a statement such as "if wp(S,R) then do S else report error." The following example demonstrates working out the full wp(S,R) calculation. Doing the full calculation for most programs is too tedious. Usually, contingency error-trap conditions are derived through a combination of formal wp calculations in easy code segments and heuristic estimates of what the calculations should look like in more difficult code segments. (See http://www.coastalresearch.com/ for definitions and more examples of both the formal and the heuristic techniques.)

In this example, the programmer interpreted the statement "the C compiler made by company X is Y2K compliant" to mean that any software compiled using that compiler would be compliant. The derived wp(S,R), if checked before the routine S executes, provides an error detector to guard against the programmer's misinterpretation.

Let S =

```
"       #include <stdio.h>
        #include <time.h>
        #include <stdlib.h>

        int maintenancedue (int lastservice)
        {
         int maintenance, servicedue, currentyear
         long currentdate;
         struct tm *t;

         time(&currentdate);
         t = localtime(&currentdate);
         currentyear = (int) t->tm_year;

①        servicedue = 5 + lastservice;
②        if (currentyear >= servicedue)
③        maintenance = currentyear;
         else
④        maintenance = servicedue;
⑤        return maintenance;
        }"
```

For a more detailed explanation of the mathematics that follow, see http://www.coastalresearch.com/. Note that "$\vee$" means "OR" or "union," "$\wedge$" means "AND" or "intersection," "$\Rightarrow$" represents "implies," "F" is false, and "T" is true. The calculation uses the fact that if the symbol "a" may be expressed as the sequence of symbols "b;c", then wp(a,R) = wp(b;c,R) = wp(b,wp(c,R)).

R = "currentyear $\leq$ maintenancedue = max(currentyear,lastservice+5) $\leq$ currentyear+5," because the "business rules" (known to the source code maintainers or users) say that maintenance occurs every five years at the maximum. The rules also say how the current year and the maintenance year are related.

wp(S,R) = wp(①;②;⑤,R) =

wp(①;②,wp(⑤,R)) =

wp(①;②,wp(maintenancedue = maintenance, currentyear $\leq$ maintenancedue = max(currentyear,lastservice+5) $\leq$ currentyear+5)) = wp(①;②, currentyear $\leq$

"Calculating Weakest Precondition for a Simple C Routine").

- Add source code to beginning or end of each subroutine (or whatever level valid input and output sets were determined) to check incoming (or outgoing) data to verify that it is "within range." If the input (or output) data is out of range, take whatever action needed, e.g., issue warning messages, write an error log, or halt execution.

Input/output validation is a good practice that may even provide a defense against the most pernicious Y2K error—unrecognized data corruption just short of system failure.

Event and Data Auditing

Audit information records when and how events occur—critical information in planning and executing recovery. Event auditing logs the actions and logs calls by routines, calls to routines, the order in which calculations take place, etc., to localize a problem's cause. Data auditing records the transactions against each data item. This traces where data corruption exists, how far it has spread, and perhaps what might be done to correct it. Together, event and data auditing diagnose a system by identifying an LE's cause and effect [6]. A specific contingency plan explains how to collect the information and what to do with it.

Using Debuggers and Source Code Instrumentation to Build "Audit" Trails

Running an "instrumented" or "debug" version of a system accumulates audit-like information to find and diagnose errors after they have occurred. Although collecting information by using test or debug tools is not traditional auditing, these tools are practical because standard audit trails are often not detailed enough. For instance, test instrumentation tools add code to an application that records every execution branch choice and perhaps even value settings. (Some debuggers have a trace mode that provides equivalent information.) Likewise, debugging tools allow "break points" to be set in software. Program execution is suspended when a break point is encountered, allowing queries on variable values. Some debuggers permit interaction; other debuggers and all instrumentation tools provide only post-execution information.

Presuming there will be no bugs from debug or instrumentation interaction with the source code at compile time, you can use the following procedure of instrumented and debug source code versions to track down and diagnose Y2K events

- Create an instrumented version of the application. Create a debug version of the application.
- Retain a non-debug, noninstrumented version of the application for execution during all periods when Y2K events are unlikely. (Debugging and instrumentation data generation may slow the application.)

maintenance = max(currentyear, lastservice+5) ≤ currentyear+5) = wp(①, wp(②, currentyear ≤ maintenance = max(currentyear, lastservice+5) ≤ currentyear+5))

However, wp(②, currentyear ≤ maintenance = max(currentyear, lastservice+5) ≤ currentyear+5) = [(currentyear ≥ servicedue) ∨ (currentyear < servicedue)] ∧ [(currentyear ≥ servicedue) ⟹ wp(maintenance = currentyear, currentyear ≤ maintenance = max(currentyear, lastservice+5) ≤ currentyear+5)] ∧ [(currentyear < servicedue) ⟹ wp(maintenance = servicedue, currentyear ≤ maintenance = max(currentyear, lastservice+5) ≤ currentyear+5)] = [T] ∧ [(currentyear ≥ servicedue) ⟹ (currentyear ≤ currentyear = max(currentyear, lastservice+5) ≤ currentyear+5)] ∧ [(currentyear < servicedue) ⟹ (currentyear ≤ servicedue = max(currentyear, lastservice+5) ≤ currentyear+5)] = [(currentyear ≥ servicedue) ⟹ (currentyear ≤ currentyear = max(currentyear, lastservice+5) ≤ currentyear+5)] ∧ [(currentyear < servicedue) ⟹ (currentyear ≤ servicedue = max(currentyear, lastservice+5)≤currentyear+5)]=R'

wp(①, R') = wp(servicedue = 5 + lastservice, [(currentyear ≥ servicedue) ⟹ (currentyear ≤ currentyear = max(currentyear, lastservice+5) ≤ currentyear+5)] ∧ [(currentyear < servicedue) ⟹ (currentyear ≤ servicedue = max(currentyear, lastservice+5) ≤ currentyear+5)]) = [(currentyear ≥ 5 + lastservice) ⟹ (currentyear ≤ currentyear = max(currentyear, lastservice+5) ≤ currentyear+5)] ∧ [(currentyear < 5 + lastservice) ⟹ (currentyear ≤ 5 + lastservice = max(currentyear, lastservice+5) ≤ currentyear+5)]

If S executes only when "[(currentyear ≥ 5 + lastservice) ⟹ (currentyear ≤ currentyear = max(currentyear, lastservice+5) ≤ currentyear+5)] ∧ [(currentyear < 5 + lastservice) ⟹ (currentyear ≤ 5 + lastservice = max(currentyear, lastservice+5) ≤ currentyear+5)]" (the result of the wp(S,R) calculation above) is true, R will be satisfied. (To calculate the truth or falseness of this quantity, note that for two symbols "a" and "b," "a ⟹ b" is false only if "a" is true and "b" is false.) Inputs that falsify the result of the calculation are error conditions and should be guarded against.

To see how this works in practice, suppose that it is now the new millennium. Let the current year be 2001, represented as currentyear=101 in the source code (check the C definitions). Supposing the last maintenance was in 1995, represented as lastservice = 1995, S might be supplied by a database to the routine S. Executing S should return that the maintenance is due now, in 2001. Instead, the routine returns 2000.

A follow-on routine that flags equipment for maintenance by checking to see if the maintenance due date is *equal* to the current year would return false–and continue to return false forever. To a casual reader (or to a hurried Y2K analyst under pressure to get the job done), the code looks fine. But if the result of the wp(S,R) calculation is checked, it will be found to be false. Using the wp(S,R) as a "guard" before executing S would prevent this error.

As a practical matter in this example, an experienced C programmer could have found the error with less work than calculating wp(S,R). In more complicated routines, or where someone with enough time and experience to read the code is unavailable, calculating wp(S,R) using a combination of formal and heuristic techniques holds the advantage.

- On any date or time when a Y2K event is likely, use the instrumented version of the application in place of the standard version. If a Y2K error appears, use the "audit" information from the instrumented version to identify the instruction sequence that occurred during the erroneous run.
- Use the debug application version to investigate the erroneous execution sequence found with the instrumented application.

Specific tools are not necessary to gather audit-like information. Anyone who can write and compile code can add debug and trace statements. Tools merely make life easier. In any case, audit information is valuable in following an event's cause and effect.

Audit Through Application Models

Some Y2K assessment and repair tools construct comprehensive source code models through reverse engineering techniques. During assessment, they help the programmer locate potential Y2K problems and may even provide insight to fix problems. To audit using these models simply means that if a Y2K event occurs, the model, rather than being used just as an error predictor, can help diagnose the problem. The model constitutes a "holistic" audit in its ability to localize problems given the real-life information about what happens when an error occurs.

Presuming that an up-to-date model of the source code and its interactions and dependencies can be maintained, a model can be used to rapidly diagnose errors as follows:

- Use a reverse engineering (see sidebar "Reverse Engineering") tool to model the application, beginning with the implementation level and working toward a design-level understanding. Identify all real and suspected date data usage. (Deriving this information is essentially what goes on during a sophisticated Y2K bug search.) Keep this model current throughout revisions. If possible, trace the user's experience-based "business rules" understanding through the application. This trace may help in understanding the side-effects of Y2K problems.
- When a Y2K bug occurs, note the functional area and as much other information as possible about the bug. Pinpoint the bug in the reverse engineered model.
- Using the model's control and data flow information, along with the data usage information, trace the bug manifestation to the code and data flows that caused the bug.
- Follow the bug as it propagates through the code and data to find all bug implications.
- Repair all problems caused by the bug. Update the model to reflect the repairs.

Generic LE Contingency Plans

Limited information about systems in Category 1 confines generic LE contingency plans to addressing broad possibilities. Generic plans end up being like physical event plans; however, generic plan techniques apply equally to systems in Categories 1 and 2. Generic contingency plan mechanisms include service degradation, internal recovery, commercial recovery, cooperative recovery, and combination recovery strategies [7].

Service Degradation Strategies

Service degradation strategies are useful when IT is partially operational. Service degradation involves

- Reduction of Service: Some, but not all, functionality is available [7]. This strategy works when part of the system experiences problems, but a work-around bridges the gap. The work-around might not be desirable or meet all requirements, but it allows something like business-as-usual pending repairs. For instance, when a central calendar management system is inoperable, anyone planning events is inconvenienced. Replacing the calendar system with a temporary text file to share information may suffice.
- Manual Replacement of IT-Based Service: IT tasks can sometimes be performed manually. This occurs when manual calculations substitute for automated functions, or paper records replace on-line data. Knowledge about the manual procedures often exists, since the IT service superseded the original manual procedures [7]. Manual procedures do not mean abandoning automation—if a corporate accounting program on a mainframe is unavailable, perhaps a personal computer spreadsheet could substitute.
- Withdrawal of Service: Functions without immediate operational impact (planning, research and development, etc.) are dispensable during an LE. The functionality may be too complex, require too much precision, or be too time-consuming for manual execution [7]. Because the functionality cannot be acceptably executed without the unavailable system and the functionality is not immediately critical, withdrawal of the service is the best choice.

Reverse Engineering

Reverse engineering is defined as "the process of analyzing a subject system to (1) identify the system's components and their interrelationships and (2) create representations of the system in another form or at a higher level of abstraction." [1] (See Figure 1.)

Reverse engineering covers a variety of techniques; only a few are relevant here. Beginning with source code, reverse engineering can, for example, produce control and data flow information (often represented graphically) both between and within routines, identify the ripple effects of changing one piece of source code with respect to other code, and even deduce the domain of valid inputs and outputs.

Reverse engineering depends heavily on automated assistance. Many fine research tools are free (see http://gulf.uvic.ca/~kenw/toolsdir/). Building and maintaining a source code date usage model using a reverse engineering tool can have a significant payoff not just in diagnosing Y2K problems before critical dates occur but after supposedly corrected problems crop up as well.

Internal Recovery Strategies

Contingency plans frequently use internal recovery strategies. These strategies feature a "can do" attitude of "the pressure is on—let's get the job down *now*!"

- Work Round-the-Clock: Working extra hard sometimes gets a job done quickly and well. However, experience teaches that work produced under pressure is often poorly done. Plans that use this strategy need to provide details on getting the best from people in a short time, maintaining morale under pressure, and choosing the right people.
- Train and Assign Extra People: Any project might benefit from extra hands. Unfortunately, a crisis is not the time to bring on new people. There may be exceptions to this if the people are of high ability, thoroughly trained, and familiar with the work but (through some quirk of fate) are working elsewhere. This strategy requires forethought on getting good people, training them adequately, and integrating them into the team.
- Have Employees On Call: Many industries use an "oncall" strategy to have employees available during unforeseen events. Unless a spot repair solves the problem, though, calling employees in at midnight might accomplish no more than creating a bleary-eyed work force. Important details for this tactic include having a checklist of trivial repairs to attempt before calling employees in, diagnostics to determine when a problem is solvable from on-call resources, and arrangements in case of unreliable communications.
- Information Preservation: A classic "disaster" recovery aid is source code and data backup. System backups are prudent. Preserving data and program code frequently, particularly if multiple versions are retained (in case the most recent backup was done *after* an event occurred but *before* the event was noticed), ensures a baseline exists. Be sure to verify that the procedure to recover saved information works *before* it is needed. Be aware, too, that backups may be of limited use during a Y2K LE, because (1) the backup may be unreadable by the Y2K-impacted system, and (2) the backup itself might include Y2K errors.

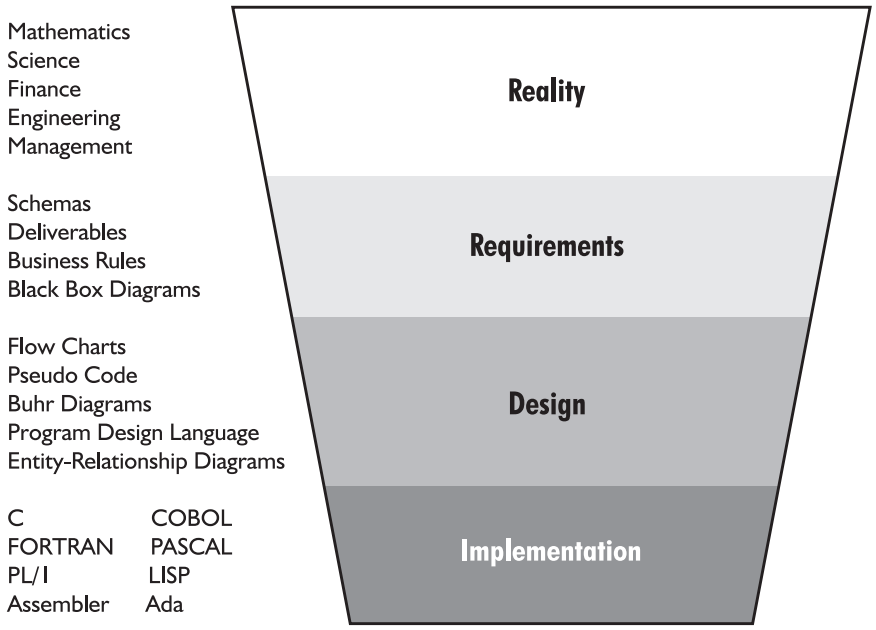


Figure 1. *Software abstraction levels.*

Commercial Recovery Strategies

Commercial recovery strategies help when an organization cannot recover from an event because of technical, personnel, or political issues but can "hire" a solution.

- Contracting Tasks to Others: Hiring supplemental employees may aid recovery from an event. Teams can temporarily expand to produce results more rapidly. Alternatively, added employees can free internal resources to concentrate on recovery. Either strategy creates difficulties similar to those in the above "train and assign extra people" solution. Important details in using this strategy include determining the types of available help and having a purchase order for services pre-approved.
- Commercially Available System Alternatives: Are there commercially available equivalents to an internally developed system? Depending on the urgency of repairs, their technical feasibility, how good a replacement the commercial alternative is for the existing component, and the cost of the repairs vs. the commercial substitute, buying a replacement is an efficient recovery method. It may require great effort to fit the commercial substitute into the existing infrastructure; so, seriously consider the impact before using this tactic.

Cooperative Recovery Strategies

Cooperation between organizations with similar systems and problems might facilitate more robust systems and more rapid post-event recovery. Banding together also gives cooperating organizations a louder voice to vendors who are making fixes and provides other opinions on how to proceed. A drawback is that cooperation helps those who did not work hard to meet the challenges while providing little benefit to those who did their homework. Cooperation also risks exposing sensitive information to potential competitors. A contingency plan that addresses this strategy helps limit the risks while enhancing the advantages by establishing nondisclosure agreements, exploring the strengths of each party, and determining administrative procedures for cooperation.

Combination Recovery Strategies

Combining the strategies above gives a more robust overall solution. For in-

stance, hiring temporary on-call employees might be as effective as using internal employees while avoiding a morale impact on long-term staff.

Summary

A good contingency plan accounts for the importance of the IT being protected, the contingency mechanisms' costs and benefits, and the ability of system developers, maintainers, and managers to implement the plan. Paramount in contingency planning is knowing a system's vulnerabilities, determining real-world threats, understanding the combination of threats and vulnerabilities, and then choosing appropriate contingency mechanisms. ◆

About the Authors

Robert L. Moore is a senior software engineer for Coastal Research and Technology, Inc. in the National Security Agency (NSA) Year 2000 Oversight Office. He is the author of Y2K compliance criteria widely used in the U.S. intelligence community and a variety of articles on software reengineering, reverse engineering, and Y2K issues. Prior to Y2K work, he worked on software reengineering projects for NSA's software engineering center. He is a certified software test engineer and has a master's of science degree in applied mathematics.

718 Meadow Field Court
Mount Airy, MD 21771
Voice: 301-688-9943
Fax: 301-688-9494
E-mail: rlmoore@romulus.ncsc.mil

Roberta H. Krupit is a senior software engineer for Coastal Research and Technology, Inc. in the National Security Agency Year 2000 Oversight Office. She has worked on software reengineering projects for NSA and the Office of Naval Intelligence.

References

1. Chikofsky, Elliot J. and James H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, January 1990, pp. 13-17.
2. Gannod, Gerald C. and Betty H. C. Cheng, "Using Informal and Formal Techniques for the Reverse Engineering of C Programs," *Proceedings of the 1996 International Conference on Software Maintenance*, IEEE Computer Society Press, Los Alamitos, Calif., 1996, pp. 265-274.
3. Gries, David, *The Science of Programming*, Springer-Verlag, New York, 1981, Chaps. 1, 9-12, 16.
4. Mayfield, Terry, J. Eric Roskos, Stephen R. Welke, and John M. Boone, "Integrity in Automated Information Systems," C Technical Report 79-91, Institute for Defense Analysis, 1991, Sections 2.1, 2.2, 3.6-3.9, 3.12. (Available by writing to INFOSEC Awareness, Attn: V/NISC, National Security Agency, 9800 Savage Road, Ft. George G. Meade, MD 20755-6753 or at http://www.radium.ncsc.mil/tpep).
5. Mohan, C., Kent Treiber, and Ron Obermarck, "Algorithms for Management of Remote Backup Data Bases for Disaster Recovery," *Proceedings of the 9th Annual International Conference on Data Engineering*, IEEE Computer Society Press, Los Alamitos, Calif., 1993, pp. 511-518.
6. National Computer Security Center, "A Guide to Understanding in Audit in Trusted Systems," National Computer Security Center, 1987, Section 5-6 (Available by writing to INFOSEC Awareness, Attn: V/NISC, National Security Agency, 9800 Savage Road, Ft. George G. Meade, MD 20755-6753 or at http://www.radium.ncsc.mil/tpep).
7. QED Information Services, Inc., *Disaster Recovery: Contingency Planning and Program Evaluation*, Chantico, Port Jefferson, New York, 1985, Chap. 4.

Notes

1. Some systems from Category 2 may be returned to Category 1 if Y2K examination reveals that the systems will be impossible to repair within existing resource constraints.
2. A useful related technique is process isolation—separating data records into two sets: one set for application X and one set for application Y to limit data-propagated errors in X from corrupting Y (and vice versa). Algorithms may be adapted from [5].
3. Least privilege or role enforcement (restricting processes to just the accesses and abilities they need for the current moment's action) is another related mechanism. For instance, if the YY part of DDMMYY increments years since 1900, YY could overflow as the counter moves from 31 December 1999 to 1 January 2000 (that is, YY = 100). To restrict any process that tries to write DDMMYY to a database to no more than six characters still allows Y2K problems to occur but prevents an accidental overwrite of adjacent data items.
4. This is true if inputs from random number generators are counted as outside inputs. Consequently, the random number is a known quantity as an input, even if it is not known until run time.

IEEE/EIA 12207 Standard for Software Lifecycle Processing

The new commercial standard IEEE/EIA 12207, "Information Technology – Software Life Cycle Processes," is available from the Defense Automated Printing Service (DAPS) at no charge. The standard comes in three parts:

- IEEE/EIA 12207.0, "Standard for Information Technology – Software Life Cycle Processes."
- IEEE/EIA 12207.1, Guide for ISO/IEC 12207, "Standard for Information Technology – Software Life Cycle Processes – Life Cycle Data."
- IEEE/EIA 12207.2, Guide for ISO/IEC 12207, "Standard for Information Technology – Software Life Cycle Processes – Implementation Considerations."

Other military and federal specifications also are available from DAPS.

Defense Automated Printing Service
Building 4/D
700 Robbins Avenue
Philadelphia, PA 19111-5094
Help Desk: 215-697-6257/6396 DSN 442-6257/6396
Fax: 215-697-1462
E-mail: roy_bowser@daps.mil
Internet: http://www.dodssp.daps.mil

Joint Service Co-Sponsors Encourage Your Attendance at STC '99

"The Eleventh Annual Software Technology Conference (STC '99), the premier Software Technology Conference in the Department of Defense, is co-sponsored by the United States Air Force, Army, Navy, and Marine Corps, and the Defense Information Systems Agency (DISA). Utah State University Extension is the conference non-federal co-sponsor. We anticipate more than 3,500 participants from the services, other government agencies, contractors, industry, and academia.

"The theme for the Eleventh Annual Software Technology Conference is 'Software and Systems for the Next Millennium.' Information used for the next millennium will require systems and software interoperability. This interoperability must be joint—across all services and forces fighting together.

"The conference theme helps focus on the need for planning to meet the critical information needs in the new millennium. A technically prepared 21st century Department of Defense demands shared information and a common view of the battlefield. This can only happen with integrated systems. The challenge is tough; the outcome will be information dominance.

"We strongly encourage your participation in STC '99. Mark your calendars for May 2-6, 1999 to be in Salt Lake City, Utah to explore new software ideas and trends. At this premier conference, government, industry, and academia software experts will explore new ideas and technologies for information systems that will be used to usher us into the next millennium."

—*Conference sponsors* Lt. Gen. David J. Kelley*, director (DISA);* Lt. Gen. William Campbell*, director of information systems for command, control, communications, and computers (U.S. Army);* Dr. Helmut Hellwig*, deputy assistant secretary of the Air Force for science, technology, and engineering (U.S. Air Force);* Rear Adm. Kenneth D. Slaght*, chief engineer, Space and Naval Warfare Center (U.S. Navy); and* Brig. Gen. Robert Shea*, assistant chief of staff for command, control, communications, computers, and intelligence (U.S. Marine Corps).*

The official STC '99 registration brochure was mailed in early January. This year, it is easier to register early. Send in your registration forms with your credit card number now, and it will not be charged until March 30, 1999. You no longer have to wait until the last minute to register.

You may visit our Web site at http://www.stc-online.org for current conference information, exhibit information, registration forms, and housing information. If you would like a copy of the registration brochure sent to you, please send an E-mail request to wadel@software.hill.af.mil or call 435-797-0039.

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

Dana Dovenbarger, Conference Manager
Lynne Wade, Assistant Conference Manager

Software Technology Support Center
OO-ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
Voice: 801-777-7411 DSN 777-7411
Voice: 801-777-9828 DSN 777-9828
Fax: 801-775-4932 DSN 775-4932
E-mail: dovenbad@software.hill.af.mil
E-mail: wadel@software.hill.af.mil

Publications on Configuration Management

CROSSTALK Articles

- Mehlman, Lon, "Implementing a Paperless Environment: The NAVSTAR GPS Block IIF Engineering Management System Project," March 1998.
- Alder, Reuel, "From the Publisher: 'Today's Software Complexity Demands Good CM,'" February 1998.
- Gill, Ted, "Stop-Gap Configuration Management," February 1998.
- Ventimiglia, Bob, "Effective Software Configuration Management," February 1998.
- "Worth a Look: Configuration Management Readings," February 1998.
- "Configuration Management Web Sites," February 1998.
- van der Hoek, André, Richard S. Hall, Antonio Carzaniga, Dennis Heimbigner, and Alexander L. Wolf, "Software Deployment: Extending Configuration Management Support into the Field," February 1998.
- Burton, Tom, "Software Configuration Management Helps Solve Year 2000 Change Integration Obstacles," January 1998.
- Starbuck, Ronald A., "Software Configuration Management: Don't Buy a Tool First," November 1997.
- Haque, Tani, "The F-16 Software Test Station Program: A Success Story in Process Configuration Management," November 1997.
- Haque, Tani, "Process-Based Configuration Management: The Way to Go to Avoid Costly Product Recalls," April 1997.
- Dart, Susan A., "Achieving the Best Possible Configuration Management Solution," September 1996.
- Haque, Sohail, "Introducing Process into Configuration Management," June 1996.
- Starbuck, Ronald A., "Software Configuration Management by MIL-STD-498," June 1996.
- Kingsbury, Julie, "Adopting SCM Technology," March 1996.
- Mosley, Vicky, Frank Brewer, Rita Heacock, Phil Johnson, Gary LaBarre, Vince Mazz, and Tami Smith, "Software Configuration Management Tools: Getting Bigger, Better, and Bolder," January1996.
- Berlack, H. Ronald, "Evaluation and Selection of Automated Configuration Management Tools," November/December 1995.
- Marshall, Alexa, "Software Configuration Management: Function or Discipline?" October 1995.
- Marshall, A. J., "Demystifying Software Configuration Management," May 1995.
- Meiser, Kenneth, "Terms in Transition: Software Configuration Management Terminology," January 1995.
- Sorensen, Reed, "Document Management Awareness Is Increasing," February 1994.

Books

- Burrows, Clive and Ian Wesley, *Ovum Evaluates: Configuration Management*, Ovum, London, 1998, ISBN 1-898-97224-9, http://www.ovum.com.
- Conradi, Reidar O., ed., *Software Configuration Management: ICSE '97 SCM-7 Workshop, Boston, Mass., USA, May 18-19, 1997: Proceedings*, Springer, Berlin, 1997, ISBN: 3-540-63014-7.
- Mikkelsen, Tim and Suzanne Pherigo, *Practical Software Configuration Management: The Latenight Developer's Handbook*, Prentice-Hall PTR, Upper Saddle River, N.J., 1997, ISBN: 0-132-40854-6.
- Buckley, Fletcher J., *Implementing Configuration Management: Hardware, Software, and Firmware*, IEEE Computer Society Press, Los Alamitos, Calif., 1996, ISBN: 0-818-67186-6.
- Burrows, Clive, George W. George, and Susan Dart, *Ovum Evaluates: Configuration Management,* Ovum, London, 1996, ISBN 1-898-97276-1, http://www.ovum.com.
- Sommerville, Ian, ed., *Software Configuration Management: ICSE '96 SCM-6 Workshop, Berlin, Germany, March 25-26, 1996 : Selected Papers*, Springer, Berlin, 1996, ISBN: 3-540-61964-X.
- Estublier, Jacky, ed., *Software Configuration Management: ICSE SCM-4 and SCM-5 Workshops: Selected Papers*, Springer, Berlin, 1995, ISBN: 3-540-60578-9.
- Kelly, Marion V., *Configuration Management: The Changing Image*, McGraw-Hill, New York, 1995, ISBN: 0-077-07977-9.
- Ben-Menachem, Mordechai, *Software Configuration Management Guidebook*, McGraw-Hill, New York, 1994, ISBN: 0-077-09013-6.
- Compton, Stephen B. and Guy R. Conner, *Configuration Management for Software*, Van Nostrand Reinhold, New York, 1994, ISBN: 0-442-01746-4.
- Ayer, Steve and Frank S. Patrinostro, *Software Configuration Management: Identification, Accounting, Control, and Management*, McGraw-Hill, New York, 1992, ISBN: 0-070-02603-3.
- Berlack, H. Ronald, *Software Configuration Management*, Wiley, New York, 1991, ISBN: 0-471-53049-2.
- Whitgift, David, *Methods and Tools for Software Configuration Management*, J. Wiley, Chichester, England 1991, ISBN: 0-471-92940-9.
- Babich, Wayne A., *Software Configuration Management: Coordination for Team Productivity,* Addison-Wesley, Reading, Mass., 1986, ISBN: 0-201-10161-0.

CrossTALK

Sponsor	***Lt. Col. Joe Jarzombek*** 801-777-2435 DSN 777-2435 jarzombj@software.hill.af.mil
Publisher	***Reuel S. Alder*** 801-777-2550 DSN 777-2550 publisher@stsc1.hill.af.mil
Managing Editor	***Tracy Stauder*** 801-775-5746 DSN 775-5746 managing_editor@stsc1.hill.af.mil
Senior Editor	***Sandi Gaskin*** 801-777-9722 DSN 777-9722 senior_editor@stsc1.hill.af.mil
Graphics and Design	***Kent Hepworth*** 801-775-5798 graphics@stsc1.hill.af.mil
Associate Editor	***Lorin J. May*** 801-775-5799 backtalk@stsc1.hill.af.mil
Editorial Assistant	***Bonnie May*** 801-777-8045 editorial_assistant@stsc1.hill.af.mil
Features Coordinator	***Denise Sagel*** 801-775-5555 features@stsc1.hill.af.mil
Customer Service	801-775-5555 custserv@software.hill.af.mil
Fax	801-777-8069 DSN 777-8069
STSC On-Line	http://www.stsc.hill.af.mil
***CROSSTALK* On-Line**	http://www.stsc.hill.af.mil/ Crosstalk/crostalk.html
ESIP On-Line	http://www.esip.hill.af.mil

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

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

E-mail: custserv@software.hill.af.mil
Voice: 801-775-5555
Fax: 801-777-8069 DSN 777-8069

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

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

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

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

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

STSC On-Line Services: STSC On-Line Services can be reached on the Internet. World Wide Web access is at http://www.stsc.hill.af.mil. Call 801-777-7026 or DSN 777-7026 for assistance, or E-mail to schreifr@software.hill.af.mil.

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

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

Don't Forget the Feather Boa

The current upward trend in information technology (IT) salaries is great, right? Don't make me laugh myself into a coma. Statistics show you're still *way* underpaid. Do an apples-to-apples comparison of IT salaries with an equivalent field, such as professional basketball. A representative IT salary (Jeremy Needers, Web design intern, $14,500) vs. a typical player's salary (Shaquille O'Neil, center, $17 million) shows you're earning roughly 1,172 times less than you're worth.

It's not fair. Why should the players get so much more wealth and attention? Is it because they're more interesting than you? More charismatic? More exciting? A rarer commodity? And might I add, far, far better looking?

Maybe we shouldn't go there. But have you noticed that a typical IT team is strikingly similar to a typical NBA team? You've got one or two star prima donnas supported by some solid starters, some backups who contribute on-and-off bursts of genius, and a few bench-warmers who do little else but fill a roster spot.

Unlike the IT world, however, an NBA team's stars make *several times* the salary of the bench-warmers. If you're sick of watching your team's desk-warmer sneak out early in a nicer car than yours, it's time to learn from the pros how to set yourself apart as your team's franchise player. Follow the principles outlined below and you'll soon be the darling of your organization, handling your superiors with the media savvy and charm of Dennis Rodman (minus the dignity).

PUBLICITY. The popular media still hasn't caught on to the thrill of watching IT's design and coding all-stars do their magic. We can only dream they'll someday wake up and give both hoop and engineering heroes equal billing:

"Van Horn Scores 38 in 2OT Nail-biter!"

"Pippen's 40-Foot Buzzer Beater Lifts Bulls over Knicks!"

"Smith's Design Review Finds Two Medium-Impact Errors! And He's Almost on Schedule!"

But the media remains out of touch—*everyone* knows Pippen plays for Houston now. Before IT gets better press, we'll have to equip offices with locker rooms where reporters can gather for an IT hero's scintillating post-milestone analysis:

"Y'know, we just went out there and programmed hard and let the algorithms come to us. We also did good following our game plan, y'know, and got some good coding off the bench. I think we just wanted it more. But I'm sick of carrying this project. I want $15 million or a trade to a project that appreciates my abilities."

Until this happens, you must use your organization's own media. For example, insert subliminal self-promotion material into your E-mails. With skill, you can humbly, subtly position yourself as the key to all past and future successes:

"To management: Before I discuss our weekly report, I must respond to the praise that has been violently heaped upon me for my performance on the Foomber project, including praise from direct competitors who are now offering exorbitant sums for my talent and insider knowledge. In good conscience, I must humbly and publicly acknowledge that I couldn't have single-handedly turned the project into the gleaming, profitable organizational ensign it is today–in contrast to the fetid pool of yak sputum it was when I arrived–without the help of my well-intentioned but far less talented co-team members. For example, Ralph Nerfderder loaned me a sharpened pencil on several occasions …"

ENDORSEMENTS. Endorsement contracts give you the leverage of appearing popular and desirable. Endorse products associated with developers. (And I'll resist the temptation to take a jab at developers by dredging up worn-out stereotypes. For example, I'd never suggest an ad with copy like, "Curaid-brand strips! They hold my glasses together twice as long as the leading brand!")

Instead, you could endorse something sexier, like pizza. Not that you'd need an endorsement contract with a pizza chain—you just need your managers to think you've got that kind of star power. If you're the computer whiz you claim to be, you should have no problem breaking into the local rag's ad department computer and modifying an ad to show a picture of yourself alone at your desk late at night, along with a snappy, subliminally self-promoting headline:

"When staying late at night redoing work done by bozos,
I make mission-critical, non-long-distance calls to Dominos!"

Use these techniques in the weeks before your next review cycle—who knows but that they'll help you get the additional $1 million or $2 million per year you desire? Have Dennis Rodman's promoter visit your office and give additional pointers. Be sure to warn him about the yak spit in the hallway. – Lorin May

Got an idea for BACKTALK? Send an E-mail to backtalk@stsc1.hill.af.mil