Did You Say CMVC?

Document Number GG24-4178-00

September 1994

International Technical Support Organization
San Jose Center

Take Note!

Before using this information and the product it supports, be sure to read the general information under "Special Notices" on page xv.

First Edition (September 1994)

This edition applies to Version 2, Release 1, Modification Level 0, of IBM Configuration Management and Version Control/6000, Program Number 5765-207, for use with the AIX Operating System 3.2

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. 471 Building 70B
5600 Cottle Road
San Jose, California 95193-0001

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Abstract

This document describes the use of the IBM Configuration Management and Version Control (CMVC) product in the scope of the downsizing of a DATABASE 2 business application from an IBM mainframe running MVS tc the RISC System/6000 and AIX/6000.

The book gives you a general view of the CMVC features and an understanding of how CMVC can be used to improve quality and increase productivity.

This document was written for customers and system engineers who need to know how to set up, configure, customize, and use IBM CMVC for UNIX in the scope of a given application development.

AD AX (245 pages)

Chapter 1. Software Configuration Management and Change Management Overview

The elements produced during the development of an application are created progressively, as new requirements are discovered and old ones are refined. You can easily forget why and when an individual element was created. You can have the latest application development tools, a highly skilled and well-managed development organization, and be following a superior development methodology, but still find that your "as-built" application does not work as it was designed, coded, tested, or documented.

It is possible that the application, which worked so well in testing, cannot be successfully re-created for delivery simply because some fixes to the code were not integrated in the final build of the application. It is also likely that some unauthorized fixes managed to find their way into the application, creating a mismatch in interfaces, calls to nonexistent subroutines, or inappropriate access to data, which no one can seem to explain. Problems of this sort represent failures in software configuration management (SCM) and change management.

Having all the right parts does not ensure a successful outcome in software development, as shown in Figure 1. It takes SCM to ensure that all the right parts are put together in the right manner, and it takes change management to ensure that any changes to those parts or their relationships are well thought out and deliberately applied.

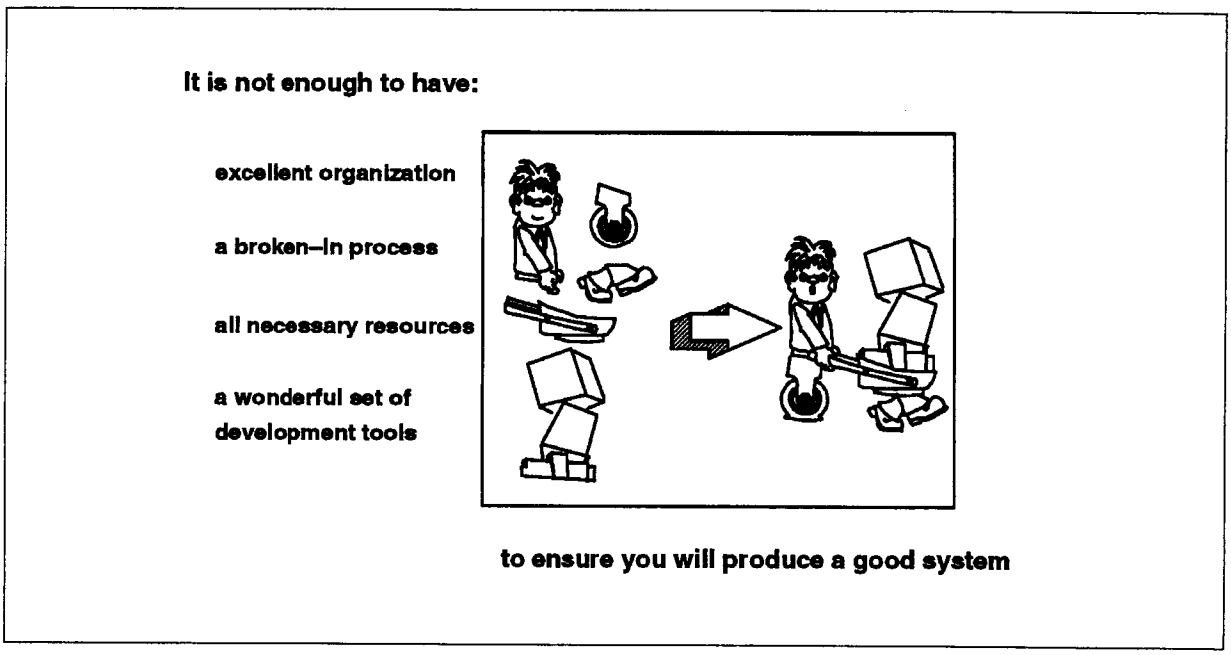

Figure 1. Why SCM and Change Management Are Necessary

In this chapter, we briefly describe the objectives and benefits associated with the processes known as SCM and change management. We also discuss the relationships among these processes, and other processes, such as project management. We also look at the relationship among these processes and software development methodologies.

1.1 Objectives

The main purpose of both SCM and change management is to ensure consistency of the elements comprising the application, as well as between the application and the documentation, which defines and supports it. The documentation that defines an application includes:

- Requirements specifications
- Interface specifications
- Design data, engineering drawings, and design specifications.

The documentation that supports an application includes various end-user manuals and separately cataloged help information.

Another very important function is to precisely identify an application's significant development "baselines." These baselines are usually associated with a major project milestone event. Typical baselines include:

- Requirements analysis and specification
- Approved design documentation
- Evaluation prototype (alpha release)
- First integration test release (beta release)
- First end-user release (first customer ship)
- Site-specific or platform-specific releases.

Identifying all the changes to a given baseline, and ensuring that they are incorporated into the next, newly forming baseline in an orderly and controlled manner is the core responsibility of change management. Change management identifies and tracks problems as well as suggested enhancements to an application. This ensures each change is carefully evaluated, and—if approved—correctly implemented and incorporated into the application.

1.2 Benefits

SCM and change management are put into place to prevent or cure problems that contribute to high development and maintenance costs, missed schedules, and customer dissatisfaction. You reap many benefits when you can successfully apply SCM and change management to software development efforts.

The primary benefit of SCM ensures that you can define and identify all elements comprising your application. It also ensures that you know exactly in which manner they are generated, preprocessed, compiled, linked or otherwise combined to form specific releases of your application and related documentation. SCM ensures that you keep a historical record of these release configurations along with the exact versions of all the components and the application itself at each release. This means that you can re-create exactly any previous release of your application, which exhibits a failing characteristic reported by end users.

SCM ensures that you can generate different parallel releases of the same application, because you know exactly which elements are unique to each version and which are common. You have a complete history of every version of every source or data file comprising your application, and you also have the ability to have parallel versions of one file incorporated in multiple releases. Release management, which is an element of SCM, ensures that you can

1.4 Automated Support for SCM and Change Management

While procedures and practices to implement SCM and change management can be manual, and in fact were for many years, they lend themselves particularly well to automation. As application development became increasingly complex, it became not only more convenient, but absolutely necessary to automate most of the tasks and procedures supporting SCM and change management. When it became possible for SCM and change management tools to take advantage of relational database technology, data about the configuration objects and change reports could be accumulated and accessed in a variety of ways beyond that necessary merely for SCM and change management purposes if the SCM tools.

When IBM undertook to develop its own UNIX-based operating system, AIX, it discovered it needed a UNIX-based industrial strength SCM and change management system that could support thousands of users, hundreds of Gigabytes of project data, and tens of thousands of report queries daily. IBM needed a product which supported its development methodology, met its QA requirements, and was compatible with its development environment. IBM needed reliability, flexibility, and performance. No one SCM product at that time included all the features, which IBM knew it required for AIX development. Many provided version control or release management, but none integrated automated change management with automated SCM

IBM, therefore, developed its own SCM and change management tool on AIX. This tool, developed for internal use, was called Orbit. After Orbit had been successfully used to bring out several releases of AIX, IBM realized that other software engineering and business application developers could also benefit from a tool with Orbit's capabilities. So, they developed a commercial SCM and change management tool from Orbit and named it Configuration Management Version Control (CMVC).

1.4.1 Configuration Management Version Control

This section gives a brief overview of the features and functions of CMVC.

CMVC is a client-server application. CMVC products execute on the HP-UX** from Hewlett-Packard** (HP**), on SunOS** and Solaris** from Sun** and on AIX/6000. Client portions of these products interoperate with any server portion. There are a command-line client, a stand-alone graphical client, and graphical client, which can be integrated into the IBM Software Development Environment (SDE) WorkBench/6000 or the HP SoftBench** environment. The CMVC server accesses data stored on its host's file system and data stored in a relational database, managed by DATABASE 2/6000 (DB2/6000*), ORACLE**, INFORMIX**, or SYBASE** products. CMVC provides a wide range of functions.

1.4.1.1 Configuration Management

CMVC provides mechanisms for identifying, monitoring, and managing changes made to a software baseline. The baseline may contain any type of data, including: documentation, design and specification data, and build and compile control information, as well as the source code itself. Files managed may contain text or binary data. CMVC supports files containing these types of data by associating them with CMVC "components." Components may be organized

into a component hierarchy to reflect the application's design, responsibilities in the development organization, or other relevant schema. Components are owned and manipulated by CMVC user IDs which are mapped to operating system user IDs, on specific network hosts.

1.4.1.2 Version Control

Version control is provided by standard UNIX Source Code Control System (SCCS), or by PVCS Version Manager**, a product available from INTERSOLV, Inc. Version control ensures that any given version of a file from the present back to its initial version can be identified and retrieved, and that the differences between any two versions can be readily identified. Version control in CMVC applies to both ASCII and binary data files.

1.4.1.3 File Change Control and History

CMVC ensures that an audit trail is maintained for every file by identifying for any file change: when the change occurred, who was responsible, and why the file was modified. If problem tracking is in place, CMVC ensures that all file changes identify the authorizing defect or feature, and that no file changes are allowed without such authorization.

1.4.1.4 Integrated Problem Tracking

Problem tracking, both for feature and defect changes is provided by CMVC. Features and defects are associated with a CMVC component. In addition to describing the enhancement proposed or problem encountered, they identify the specific versions of all controlled files, which implement the feature or defect. Problem tracking implements a configurable process. This means that defects and feature processing can be omitted. If they are used, defect and feature processing can go through a series of states, some of which are optional. Defects and features can be opened, cancelled, returned, or implemented after an optional design, size, and review subprocess is conducted. There is also an optional verify subprocess to verify that the changes were satisfactorily incorporated in a formal release.

1.4.1.5 Release Management

CMVC supports the concept of a "track," which is a mechanism to relate an individual defect or feature with the set of file changes that implement that defect or feature in a given "release" or "level of a release" of an application. Use of tracks is also a configurable process; tracks processing is optional, and if used, has optional subprocesses for approval, fix, and test. If used, tracks go through a series of states which include: approve, fix, integrate, commit, test, and complete.

Releases and levels are CMVC mechanisms for defining interim baselines of the application. CMVC records the exact version of every file comprising the release, including build instructions, and can extract those files into build directories. Release management is a configurable process which can include or omit the track process, and if the track process is employed, an optional level subprocess. A level is a group of changes that are incorporated into a release in a sequential and carefully monitored manner. A level is first in a working state, then tested in an integrated build committed when satisfactorily tested and marked as complete when all changes identified for that level have been successfully incorporated into the release.

1.4.1.6 Access Control

Components provide a mechanism by which CMVC controls access to files under its control. Access of a variety of sorts can be defined for all files associated with a given component. CMVC user IDs implicitly acquire some access authority for components by virtue of owning them, and may inherit other access authority from parent components. They can explicitly grant or deny access authority over components which they own to other CMVC user IDs.

1.4.1.7 Automatic Notification

CMVC provides for automatic notification of CMVC actions affecting particular components and their files to "interested" users. Notification is provided by electronic mail, so a user does not have to start up CMVC to be aware of the CMVC actions. A CMVC user ID's "interest" in being notified of CMVC actions can be specified in terms of specific CMVC actions and affected components.

1.4.1.8 Customization

CMVC allows additional fields to be added to the database records that implement CMVC features, defects, files, and users. These new fields are reflected by appropriate changes to CMVC windows, reports, and command-line parameters.

CMVC also enables configuration of the processes that manage CMVC objects, such as files, features, and releases. Configuring these processes determines the various states through which these objects can pass.

CMVC allows you to define "user exits" that automatically execute a UNIX shell command file, or user-written executable program whenever specific CMVC commands are executed. You are allowed to select parameter data, related to the CMVC action and object it is affecting, to be passed by the CMVC command to your the shell command file or program. You can also determine if the user exit is triggered before or after the CMVC command executes.

Chapter 4. Planning for CMVC

In this chapter we offer some general advice on planning to use CMVC, as well as offer some examples of how to apply specific features or functions in CMVC. We also show how a small application development project, described in Chapter 3, "Overview of the Application Development Project" on page 41 and alluded to in Chapter 2, "Discovering CMVC: An New Application Project Is Introduced to CMVC" on page 11, planned for its use of CMVC.

4.1 Why Plan?

The most important thing to understand about CMVC is that you do not need to understand all of it, before you begin using some of it. CMVC is broad in its function, thorough in its implementation, and very flexible. CMVC provides many mechanisms to help you accomplish your SCM goals, but it does not dictate exactly how you should use them, nor does it require that you use them all if you use only some of them. This is one of the distinguishing advantages of CMVC. Because no two development efforts have exactly the same number of requirements, the same degree of complexity, scope of effort, or the same hardware and software resources, how they approach SCM with CMVC varies significantly. Recognizing this, CMVC was designed to be set up, tailored, and utilized according to the needs of the individual project. Therefore, it is wise to plan in advance which features of CMVC you want to use initially, how you want to apply them to your SCM problems, and which features you want to phase in gradually.

The first step in this planning process is to go over the CMVC product documentation carefully. These documents identify and define CMVC objects, such as Files, Releases, and Users, and describe how CMVC users access and manipulate them. In *IBM CMVC Concepts*, you find a description of CMVC concepts, objects, processes, and interactions. Look in *IBM CMVC User's Guide* and *IBM CMVC Commands Reference* for details on how to use individual commands and GUI windows. *IBM CMVC User's Reference* provides a place to look up lists of options, record structures, and field attributes.

However, what you will not find in these documents is the *correct* interpretation of how to map CMVC objects to the real objects of your application development effort. Nor will you find advice on which circumstances call for using or not using a given CMVC object or function. This is because there is no single correct application of CMVC; this will vary with your circumstances.

The second step in planning, therefore, is to compare your thoughts of how to apply CMVC to someone else's actual experience. Since not everyone can do this, we have written this chapter. It provides a practical example of how to plan for and apply CMVC objects, concepts, and processes to meet the needs of an actual software development project. This chapter also suggests alternative approaches to CMVC that were not used on this project, but are based on other experiences with CMVC.

SCM is not a short term effort, nor does it exist in isolation. SCM responsibilities for an application begin during its development and continue as long as it is in use. The SCM effort on an individual application development project is also part of a larger SCM effort in the development organization. Therefore, original

plans for CMVC are subject to revision as the needs of the project change and as additional projects start up. Your use of CMVC will also evolve as you become more familiar with its capabilities. Generally speaking, CMVC is amenable to this fact of life. But, some decisions about CMVC that you make at the beginning of a project, have a long term impact. This book helps you distinguish between these and other decisions, which you can make tentatively now and the plan to modify as time passes.

CMVC provides a command-line interface client, as well as a GUI client. Rather than refer to specific command and parameter names, and equivalent GUI window and menu items in this chapter, we refer to CMVC actions by a generic name. For example, we refer to the *FileCreate* CMVC action, when we mean either the **File -create** command, or the **Create** selection on the Actions pull-down of the CMVC - Files window. You should refer to the manuals to identify the actual correct spelling of the command and parameter, or window and menu item names.

4.2 Pre-Installation Planning for CMVC

Before you install CMVC, you should:

- Plan your network license requirements and distribution of licenses over your hosts
- Plan your distribution of CMVC client and server software across your hosts
- Identify and define the purposes served by your CMVC families.

4.2.1 Planning Network License Requirements

CMVC makes use of Network Licensing System** (NetLS**). For details on how NetLS works and how CMVC makes use of the NetLS licensing mechanism refer to 6.2, "NetLS Installation and Initialization" on page 163. There are some decisions you must make regarding NetLS; they are primarily questions of network and system administration. These include whether to have more than one NetLS license server, and how to distribute license tokens for various licensed programs among them. The primary decision you must make regarding CMVC and NetLS, however, is how many CMVC license tokens your project will require.

4.2.1.1 What to Consider in Planning Network License Requirements

To plan your CMVC license token requirements, try to determine the maximum number of users who will be using CMVC at any one time. Someone performing SCM functions may need access all day long, while your developers may use CMVC infrequently. Not all developers will use CMVC to the same degree; it may depend upon their specialized role in your project. The project and team leaders may use CMVC a few times daily, while testers and build integrators may use it constantly.

Before a CMVC client issues a request to the server, it requests a license, or a token. After getting it, that CMVC client holds it a minimum of fifteen minutes. (This is CMVC's default minimum expiration time). If, in the next fifteen minutes, that client issues another request to the CMVC server, that token's expiration time is extended again by fifteen minutes. So, if you bring the CMVC client GUI up, make and refresh queries, display new windows, and perform CMVC actions every few minutes all day long, you will effectively use one token for most of that