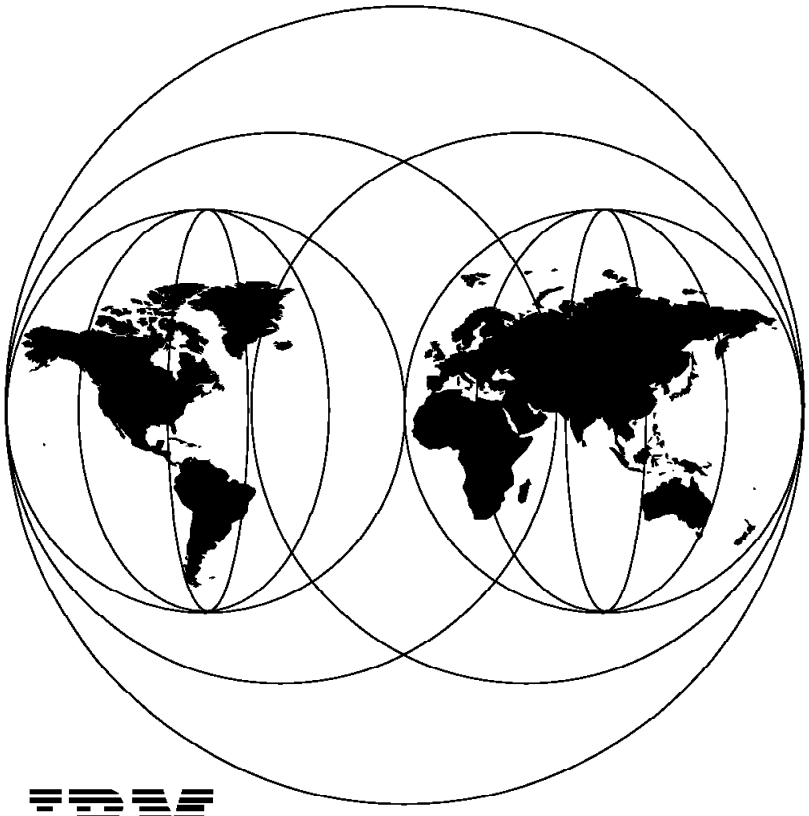
International Technical Support Organization

GG24-4345-00

Looking at CMVC from the Customer Perspective

May 1995



IBM

International Technical Support Organization

San Jose Center

International Technical Support Organization

GG24-4345-00

Looking at CMVC from the Customer Perspective

May 1995

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 (May 1995)

This edition applies to Version 2 Release 1 and Release 2 of IBM Configuration Management Version Control/6000 (Program 5765-207), IBM Configuration Management Version Control for HP systems (Program 5765-202) and IBM Configuration Management Version Control for Sun systems (Program 5622-063).

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 070B
5600 Cottle Road
San Jose, California 95193-0001

Abstract

This volume reports on a study of how CVMC is used at several IBM customer sites and IBM Laboratory installations to solve the difficult and inevitable problems associated with software configuration management in a team programming environment. This book characterizes how these “customers,” both inside and outside of IBM, used, configured, tailored and extended CMVC to meet their unique and varied requirements. This book presumes some knowledge of both CMVC and software configuration management principles. This book describes the scale-ability, applicability, and adaptability of CMVC. It provides a customer perspective on why different types and sizes of application development efforts need CMVC and what benefits they derive from its use.

(130 pages)

Contents

Figures

Tables

Special Notices

This publication is intended to help new or potential CMVC customers to envision how CMVC can be used to solve the difficult problems related to software configuration management. The information in this publication is not intended as the specification of any programming interfaces that are provided by Configuration Management Version Control/6000, Configuration Management Version Control for HP, or Configuration Management Version Control for Sun. See the PUBLICATIONS section of the IBM Programming Announcement for CMVC for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, CT 06904 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms, which are denoted by an asterisk (*) in this publication, are trademarks of the International Business Machines Corporation in the United States and/or other countries:

ADSTAR
ADMS
AIC
AIX
AIX/6000
AIXwindows
AS/400
CUA
DATABASE 2
DB2
DB2/6000
IBM
OS/2
OS/400
RETAIN
RISC System/6000
SAA

The following terms, which are denoted by a double asterisk (**) in this publication, are trademarks of other companies:

NetLS	Apollo Computer, Inc., a subsidiary of Hewlett-Packard Co.
Network Licensing System	Apollo Computer, Inc., a subsidiary of Hewlett-Packard Co.
NCS	Apollo Computer, Inc., a subsidiary of Hewlett-Packard Co.
Network Computing System	Apollo Computer, Inc., a subsidiary of Hewlett-Packard Co.
British Telecom	British Telecommunications PLC
BT Laboratories	British Telecommunications PLC
Continental	Continental AG
General Tire	Continental AG
Continental Tire	Continental AG
Uniroyal	Continental AG
Semperit	Continental AG
VAX/VMS	Digital Equipment Corp.
DEC	Digital Equipment Corp.
DEC Ultrix	Digital Equipment Corp.
Hewlett-Packard	Hewlett-Packard Company
HP	Hewlett-Packard Company
HP-UX	Hewlett-Packard Company
SoftBench	Hewlett-Packard Company
INFORMIX	Informix Software, Inc.
PVCS Version Manager	INTERSOLV, Inc.
Internet	Internet, Inc.
POSIX	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
X Window System	Massachusetts Institute of Technology
MIT	Massachusetts Institute of Technology
MCI	MCI Telecommunications Inc.
Micro Focus	Micro Focus Limited
Micro Focus COBOL	Micro Focus Limited
Toolbox	Micro Focus Limited
MS-DOS	Microsoft Corporation
Microsoft	Microsoft Corporation
Microsoft Windows	Microsoft Corporation
NetWare	Novell, Inc.
Motif	Open Software Foundation, Inc.
OSF	Open Software Foundation, Inc.
OSF/Motif	Open Software Foundation, Inc.
ORACLE	Oracle Corporation, Inc
SCS	SEIKO Communications Systems Inc.
SEIKO RECEPTOR	SEIKO Communications Systems Inc.
Messagewatch	SEIKO Communications Systems Inc.
SEIKO RECEPTOR Messagewatch	SEIKO Communications Systems Inc.
Sun	Sun Microsystems Incorporated
SunOS	Sun Microsystems Incorporated
Solaris	Sun Microsystems Incorporated
NFS	Sun Microsystems Incorporated
Network File System	Sun Microsystems Incorporated
SYBASE	Sybase, Inc.
UNIX	X/Open Company Limited

Preface

This book arises from a study of several IBM customer sites and IBM development laboratories where CMVC is used to solve the difficult and inevitable problems associated with software configuration management in a team programming environment. This book characterizes how a few specific CMVC “customers,” both inside and outside of IBM, used, configured, tailored and extended CMVC to meet their unique and varied requirements.

This document will be valuable to potential CMVC customers who are evaluating CMVC, and to current CMVC customers who are looking for new ideas on how to get the most from CMVC. The examples in this volume show some of the many ways CMVC can meet software configuration management requirements of application development efforts. This study shows, from the experienced CMVC customer perspective, how and why customers value CMVC as an essential tool in their application development environment.

How This Document is Organized

The document is organized as follows:

- Chapter 1, “Introduction to CMVC”

  This chapter gives a brief overview of CMVC and its evolution.

- Chapter 2, “Looking at CMVC through Customers' Eyes”

  This chapter describes how the authors researched this book and summarizes their findings.

- Chapter 3, “Use of CMVC on a Small Scale or in the Initial Phases of Development”

  This chapter characterizes the use of CMVC on small scale, and in the initial phases of application development.

- Chapter 4, “Use of CMVC on Medium-Scale or Companywide Basis”

  This chapter characterizes the use of CMVC on a medium-sized scale, which includes its being introduced as a company standard, its use for multiple small projects and its use on a medium-sized project.

- Chapter 5, “Use of CMVC on Very Large-Scale Basis”

  This chapter describes how CMVC is used on a very large scale in a mission critical role by a computer hardware and software vendor.

- Chapter 6, “Conclusion”

  This chapter summarizes how the customers use CMVC in various different ways and for different purposes.

- Appendix A, “Customer Profiles”

  This appendix provides a chart to summarize some key quantifiable characteristics of CMVC use at the customer sites studied.

- Appendix B, “Software Configuration Management and Change Management”

This appendix introduces the terminology and concepts of software configuration management. This material is provided as reference; it is extracted from other IBM Redbooks and White papers.

- Appendix C, “Implementation of ISO 9001 Using CMVC”

  This appendix describes how CMVC can support ISO 9000 Series requirements. This material is provided as reference; it is extracted from other IBM Redbooks and White papers.

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

- *IBM CMVC Concepts*, SC09-1633, introduces the fundamentals of the configuration management, version control, change control, and problem tracking features of CMVC. It also defines the concepts that are the foundation of the CMVC actions and describes their interrelationships.
- *IBM CMVC Server Administration and Installation*, SC09-1631, contains detailed information for planning, installing, customizing, operating, and maintaining the CMVC Server.
- *IBM CMVC Client Installation and Configuration*, SC09-1596, contains the detailed information you require to install and configure the CMVC clients.
- *IBM CMVC User's Guide*, SC09-1634, describes how to use the graphical user interface.
- *IBM CMVC User's Reference*, SC09-1597, contains the reference lists, tables, and state diagrams for CMVC, as well as a description of how message-integrated CMVC uses the Broadcast Message Server (BMS), to fully integrate with the other integrated development environment tools.
- *IBM CMVC Commands Reference*, SC09-1635, describes the syntax and usage of the CMVC commands as implemented in the command-line interface.
- *NetLS Quick Start Guide*, SC09-1661, provides the information needed to set up the Network License System (NetLS) software to work with CMVC.
- *Managing Software Products with the Network License System*, SC09-1660, provides the information needed to manage the use of NetLS software with CMVC.

International Technical Support Organization Publications

- *Did You Say, CMVC?*, GG24-4178
- *AIX Application Development: Migrating an MVS DB2 COBOL Application to AIX Using WorkBench, AIC, CMVC, etc.* , GG24-4177

A complete list of International Technical Support Organization publications, with a brief description of each, may be found in:

Bibliography of International Technical Support Organization Technical Bulletins, GG24-3070.

To get listings of redbooks online, VNET users may type:

TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG

How to Order Redbooks

IBM employees may order Redbooks and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-284-4721. Visa and Master Cards are accepted. Outside the USA, customers should contact their IBM branch office.

You may order individual books, CD-ROM collections, or customized sets, called GBOFs, which relate to specific functions of interest to you.

Acknowledgments

The advisor for this project was:

Lorna Conas,
International Technical Support Organization, San Jose Center

The book was completed and published by:

Leif Trulsson,
International Technical Support Organization, San Jose Center

The authors of this document are:

Dr. Silvana Contin,
IBM Deutschland Informationssysteme GmbH, Stuttgart, Germany

Richard Kortmann,
IBM Deutschland Entwicklung GmbH, Hannover, Germany

Maggie Cuttler, Editor
International Technical Support Organization, San Jose Center

This publication is the result of a residency conducted at the International Technical Support Organization, San Jose Center. Thanks to the following people that were being interviewed during this residency or who contributed information in other ways to this book:

Continental

- Rolf Terei, Hannover

British Telecom

- Chris Alves-Jackson, Martlesham
- Richard Cutler, Ipswich

MCI

- Craig Braze, Colorado Springs
- Melanie Havnaer, Colorado Springs
- Cheryl Herrington, Colorado Springs
- Michael Smith, Colorado Springs
- Bob Wise, Colorado Springs

IBM Austin

- Werner Hahn, Austin
- Jeri Lyn Hilsabeck, Austin
- Tom Howe, Austin
- Tom Krueger, Austin
- Dennis Lee, Austin
- Landy Ligon, Austin
- Dana Lloyd, Austin
- April Nicolay, Austin
- Bob Pietrasik, Austin
- Linda Pryer, Austin
- Marcia Proctor, Austin
- Tim Schuetze, Austin
- Conway Wharton, Austin

SCS

- Lyn Anderson, Portland
- Dick Moran, Portland
- Dennis O'Brian, Portland
- Dan Park, Portland
- Dave Ritchie, Portland
- Bruce Schroeder, Portland

IBM Tucson

- Mark Balstad, Tucson
- Chia-Hong (Kevin) Chen, Tucson
- Doug Forester, Tucson
- Brian Gross, Tucson
- Gregory Hill, Tucson
- Jonathan Kiser, Tucson
- Xinmin Ling, Tucson
- Colin Moss, Tucson
- Bill Raywinkle, Tucson
- Cheng-Chung Song, Tucson
- John VanderWerf, Tucson
- Andy Zaepfel, Tucson

Chapter 1. Introduction to CMVC

Software configuration management is the process of identifying, monitoring and managing changes to software configuration items. Configuration Management Version Control (CMVC) provides a full range of functions in support of software configuration management.

This chapter provides:

- A brief look at the evolution of CMVC from an IBM internal project used in the development of the AIX operating system
- A brief overview of the product
- An introduction to the main CMVC concepts and terminology.

1.1 Evolution of CMVC

This section describes how CMVC emerged from an internal development tool to become an industrial-strength commercial product. The history of CMVC mirrors what happens in many companies as the importance and complexity of their software configuration management problems grow. Application development organizations, finding their choice of commercial software configuration management products inadequate, often attempt to build configuration (CM) tools themselves. Homegrown CM tools, however, become mission critical very quickly. Rarely are they designed and implemented with sufficient foresight to meet the growing needs of the application development group. The resulting tool maintenance efforts consume growing resources and distract the application development organization from its primary development efforts.

1.1.1 Hybrid AIX and Mainframe CM Solution

IBM's RISC System Division AIX operating system developers originally developed CMVC to enable them to maintain and concurrently develop versions of AIX-related system and application software products.

In the early days of AIX development (1989), IBM developers used a configuration management tool called OPATS hosted on an IBM mainframe to track program defects and a combination of SCCS and shell programs on the IBM RT/PC computer to do version control and release management and build automation. This latter collection of tools was called PLIST.

OPATS was rich in function and could handle large amounts of data with good performance because it was based on a mainframe database product. It suffered the disadvantage, however, of not being integrated with the version control and release management and build tools on the RT. Thus, while those tools were ported to AIX, OPATS remained on the mainframe. This lack of protability continued to present problems to the AIX development team.

1.1.2 Commercial Solutions Found Wanting

When the AIX developers looked for a UNIX-based commercial product to automate software configuration management, they could not find a single product that incorporated all of the qualities they sought. They needed an industrial-strength product that was robust and reliable. They needed a product that could deal with their initially large volume of data, large number of users, and high rate of transactions, but they also needed a product that could continue to scale up at the anticipated rates of growth. They needed a sophisticated problem-tracking mechanism that could conform to and reflect their evolving development process. They needed to control and manage the development of software by large numbers of independent and geographically distant groups of developers. They also wanted the CM tool to be UNIX-based, and if possible, AIX-based.

1.1.3 Write a Quick and Easy CM Tool

Because the AIX developers did not find a suitable commercial CM tool, they undertook to develop a tool themselves. The tool they developed, NEDS, was neither sophisticated in its design nor efficient in its execution. It went through two iterations before evolving into the precursor of CMVC Version 1. NEDS I was written largely in Korn Shell scripts and used SCCS for version control over source files. NEDS I stored configuration management data in large and unwieldy flat files. The AIX developers continued to use OPATS for defect tracking.

1.1.4 Rewrite for Performance Improvements

The second iteration, NEDS II, was reimplemented in C for performance. It took advantage of a commercial relational database management system (RDBMS) product to replace the flat files in which configuration management data had been stored. Performance benefits accrued from the use of the rapidly improving relational database technology. This implementation still suffered the disadvantage of not having defect tracking on a compatible platform, however. Over time, NEDS II also became inadequate to meet the growing demands of AIX development.

1.1.5 Commercial Offerings Still Lacking

Before beginning their third iteration of NEDS, the AIX developers again looked at the “make versus buy” decision. Willing to purchase a solution—even one that ran on a competitor's UNIX platform, if necessary—they considered solutions for both Sun and Apollo computers. These commercial solutions still lacked the integration of problem tracking with version control and release management, and they did not take advantage of an RDBMS.

The AIX developers were beginning to discover the benefits of access to configuration management data through a standard SQL interface, and they decided that giving up the RDBMS that made that possible would be a mistake. They concluded that they would have to develop a new tool by capitalizing on the experience they had acquired with NEDS I and II.

1.1.6 Orbit

Orbit was the next-generation CM tool that the AIX developers built. Carefully designed to meet both current and future needs, Orbit was, finally, a satisfactory solution.

According to one of the original developers of Orbit, the secret of CMVC's success is its adaptability and scalability. These characteristics were enabled by three key design principles:

- Client/server architecture
- Use of commercial RDBMS to store metadata and a commercial version control system to store versions of file objects
- Integration of the problem tracking mechanism with version control and release management to achieve full configuration management functionality in a single tool.

1.1.7 Explosive Demand for Orbit within IBM

By word of mouth, news of Orbit spread within IBM. Demand for copies grew within the company. Maintaining, distributing, and supporting Orbit and educating end users about the product became an unplanned responsibility of the AIX development team. The AIX developers, therefore, gave Orbit to IBM's Software Solutions Division in 1991 so they could evaluate it as a potential commercial product. The internal demand for this tool was driven by lack of comparable commercial products and by Orbit's particular success at meeting the requirements of team programming in a LAN workstation environment.

1.1.8 Release of CMVC/6000 Version 1

Orbit was turned over to IBM's Software Solutions Division in 1991. This division released a Licensed Program Product in 1992 called Configuration Management Version Control/6000 (CMVC) Version 1. This product was available only for the AIX operating system on the RISC System/6000.

PVCS was offered in addition to SCCS as the underlying version control mechanism. ORACLE RDBMS was supported initially. CMVC was integrated into IBM's Software Developer's Workbench/6000 and became a member of the AIX Application Development family of products. In addition to the command line interface, a GUI was introduced. Another feature was that customers could choose to use CMVC in either binding or nonbinding control fashion. Binding control imposed a rigid process on problem tracking and release management; nonbinding control imposed fewer restrictions and was appropriate in the early stages of development effort.

The AIX developers at IBM Austin, meanwhile, continued to use Orbit for about one year. Then they migrated to CMVC Version 1, although they continued to support the older Orbit user interfaces. They also continued to support front-end tools that they had written for Orbit, but they modified them to access the new CMVC Version 1 server.

1.1.9 CMVC Version 1 Enhancements

CMVC Version 1 was enhanced several times. Both the client and server portions of CMVC were later ported to Hewlett-Packard and Sun computers for use with their UNIX-based operating systems. Clients and servers on all platforms could interact with each other. Support for SYBASE and INFORMIX RDBMS products emerged, although not every RDBMS product was supported on all platforms.

1.1.10 CMVC Version 2

In Version 2 the CMVC GUI was enhanced with graphical representations of the component hierarchy, file history, and release/level history. Additional CMVC clients were later developed for PC workstations (OS/2 Version 2.1 and Microsoft Windows). Version 2 saw several server-end improvements, including expansion of configurable fields and enabling of many more user exits. CMVC Version 2 introduced the notion of configurable process definition for components and releases, replacing the older concepts of binding and nonbinding control. CMVC also added support for the DB2/6000 RDBMS.

1.1.11 Continued Spread of CMVC within IBM

CMVC popularity continued to grow within other divisions of IBM. Application developers in these divisions also developed local custom clients on VM, DOS, and OS/2 Version 1 and even built a client using AIX's own System Management Interface Tool (SMIT). (SMIT is both a general-purpose user interface development tool and a specialized system management feature. SMIT can be used to develop special-purpose menu-based command generators for both GUI and ASCII terminals.) These clients were not absorbed into the formal product but helped advance the use of CMVC within the various IBM development laboratories.

1.2 Key CMVC Design Points

CMVC is the embodiment of lessons learned by a large software vendor wrestling with the development of a tool to meet serious software configuration management challenges. The three key design decisions made for Orbit, carried through in CMVC, are largely responsible for CMVC's ability to scale and adapt to a wide variety of requirements. The full benefits of these design characteristics, although not all envisioned by the original architects, have made CMVC particularly valuable to its customers.

1.2.1 Client/Server Architecture

The AIX developers worked primarily on a large and complex network of independent RISC System/6000 desktop workstations, network-attached graphical terminals (Xstations), and large-capacity RISC System/6000 servers that supported both direct-attached graphical (HFT) and ASCII terminals. In this development environment, the decision to develop a client/server architecture that relied on TCP/IP and NFS, both standard features of AIX, was an easy choice. This decision, it turned out, was to have unexpected additional benefits for later CMVC customers.

One obvious benefit of a client/server architecture is that you can independently modify either client or server by adding new features, improving performance, or increasing customer flexibility in prerequisite software products so long as you keep the client/server interface unchanged. It also makes it easier to develop new

clients and/or user interfaces that use existing clients. For instance, the CMVC graphical user interface (GUI) was developed as a new client that conformed to the same client/server interface to which the original command-line-oriented client had been built.

This design also makes it easy to add network licensing to CMVC. By modifying the CMVC server to obtain network license tokens on behalf of any client requesting its services, the only serious change required in the client is to enable it to recognize the error conditions that the server might report if network tokens were not available to service a CMVC client request.

The choice to develop a UNIX-based product, in itself, ensures that CMVC can be easily ported to a wide variety OEM hardware. Such hardware need only offer either a native UNIX implementation or POSIX and TCP/IP and NFS interfaces in a non-UNIX operating system. This fact, coupled with the client/server architecture, ensures that the servers and clients can be implemented on various operating systems and hardware platforms and interoperate with each other.

Thus the CMVC developers were able to meet the requirements of IBM customers for HP, Sun, DOS, OS/2 2.1 and Windows CMVC clients, as well as for HP and Sun and AIX CMVC servers.

1.2.2 Use of Independent Relational Database and Version Control Systems

A significant CMVC design decision was to make use of RDBMSs that support standard SQL interfaces for query and update. Configuration management requires the storage and management of considerable data about who does what to the controlled software for which purposes. This data goes beyond merely keeping track of all historical versions of files to identifying which specific versions of all relevant files constitute specific releases of the larger software products or applications. This data also allows CMVC to model the development processes by enabling it to record state and status data and the relationships between the development objects. It is sometimes called *metadata* or data about the data (that is, about the controlled software).

This metadata lends itself to storage and manipulation within relational tables. The decision to remove responsibility for handling this data from CMVC and place it under control of a commercial RDBMS left CMVC developers free to concentrate on the issues directly related to automation of software configuration management. This decision also made it possible to host CMVC on top of a choice of RDBMS products. Now if one RDBMS failed to scale as needed, another could be slipped in its place to achieve the performance or capacity required.

Again, this decision had the additional benefit of allowing CMVC customers to make an initial choice of RDBMS based on such factors as preexisting licenses or system administration familiarity with one RDBMS or another. The number of RDBMSs on various platforms supported by CMVC has grown, and includes even IBM's own DB2/6000 at this time. As new RDBMS support has been introduced, CMVC has also been released with utilities to help customers migrate from one database product to another.

Choosing to use RDBMSs proved itself a very sound design decision, from another point of view too. As the demands placed upon CMVC at IBM Austin grew, the ability of commercial RDBMS products to handle ever larger amounts of data, on

ever larger UNIX-based computers at ever faster speeds of access, coincidentally evolved. This meant that CMVC did not require redesign to meet the demand for increased capacity and performance.

CMVC also supports a choice of version control mechanisms, another product feature that provides customers with flexibility. Users can use SCCS, the native UNIX version control system that is bundled with most UNIX-based operating systems. CMVC Version 2 also provides an ability to version binary objects. Customers can also choose to use Intersolv's PVCS to implement file version control. (PVCS, a commercial product that some customers may prefer, can version ASCII or binary files.)

1.2.3 Integrated Problem Tracking and Release Management

The inclusion of problem tracking with version control and release management is key to enabling full-function configuration management. Configuration management includes not only keeping versions of individual files and recording exact configurations of larger software entities such as releases but also the ability to trace the origin and approval history of all changes entering evolving software baselines.

The integration of problem tracking with release management and version control allows CMVC to force the discipline of a process model on the development effort. Initially, CMVC offered two model choices, a fairly sophisticated model, known as *binding control*, and a simple model, known as *nonbinding control*. CMVC Version 2 introduced a much more flexible model process, the *configurable process*. CMVC customers can customize CMVC to mimic and automate their existing configuration management process or create a process that had not previously existed.

1.3 Product Description and Prerequisites

CMVC is a client/server-based Licensed Program Product from IBM. CMVC products, both client and server, are available for HP, Sun, and AIX-based operating systems. CMVC client-only products are also available for Microsoft Windows and OS/2 operating systems. Client products will interoperate with any server product across a LAN using TCP/IP and Network File System (NFS).

CMVC UNIX-based clients offer a command line user interface, a stand-alone GUI, and a message-integrated GUI for use with IBM SDE WorkBench/6000 or the HP SoftBench products. The command line and graphical user interfaces are approximately equivalent in function, but each has particular advantages in different situations. The command line interface is used with the HP-UX, SunOS, Solaris, or AIX shells; it can be used from any ASCII terminal. The GUIs support OSF/Motif environments and therefore require that their output be displayed at Xstation or high function terminals.

CMVC requires one of the following products:

- SCCS, which comes with the AIX Application Development Toolkit
- Intersolv PVCS Version Manager.

CMVC also requires installation of one of the following RDBMSs:

- Oracle
- INFORMIX

- Sybase
- DB2/6000.

1.4 Identifying Configuration Items

A software configuration item is a composite object that can be decomposed into smaller composite objects. These smaller composite objects may themselves be decomposed into smaller objects recursively. Ultimately these objects decompose into atomic programming objects with which specific files containing source, binary data, executable, documentation, build instructions, and/or other data are associated. Therefore, a configuration item must be identified both as the aggregate whole and in terms of the smaller elements that comprise it.

CMVC supports the identification of configuration items, their internal composition, and the subordinate elements comprising them by allowing users to define and manipulate CMVC objects such as the CMVC family, component hierarchy, files, and releases. These concepts and their interrelationships are key to understanding how CMVC supports software configuration management.

The reader is presumed to be fairly familiar with configuration management concepts and terminology and reasons for doing software configuration management. If the reader is not, we provide a brief introduction to that topic in Appendix B, “Software Configuration Management and Change Management” on page 103.

1.4.1 CMVC Family

A CMVC family is a logical organization of related development data. The data in one family cannot be shared with the data in another family. Within each family, data is organized into groups called components.

1.4.2 CMVC Component Hierarchy

Components are arranged in a hierarchical structure with a single top component called *root*. Components are the focal point for information retrieval, access control, notification control, problem reporting, and data organization within CMVC. In the component hierarchy, any component may contain, that is, logically group, either subordinate components, or data files, although typically, files are associated only with end-node components in the hierarchy. Component hierarchies are fairly flexible, allowing for one child component to have multiple parents, and one parent to have multiple child components.

1.4.3 CMVC Files and Versions

A CMVC file is controlled and managed by one component. A CMVC file is uniquely identified by a path name, which consists of a UNIX directory path prefix and a base file name. Thus, there can be two unique CMVC files that have a different path prefix but the same base file name, because together the prefix and base parts would define a unique value.

Files may contain either binary or ASCII data. The purpose or contents of CMVC files is irrelevant to CMVC. These files are stored in a special file system, manipulated by the version control system, and accessible only indirectly through CMVC.

1.4.4 CMVC Releases

All files that make up a single version of a configuration item, are identified with a CMVC release. A single release may group files managed by a single component, or it can group files managed by an arbitrary number and selection of components.

A CMVC file cannot be created without reference to a release. If one release needs to contain a file already existing in another release, CMVC creates a link from the two releases to the current version of that file. Now, both releases are said to *share* the same file. Because each is using the same version of that file, it is called a *common* file.

During development, the contents of a CMVC file associated with a given release may change many times. Whenever a new version of this file is recorded the developer can choose to keep the current version linked into all releases in which it is common. Or, the developer can choose to break the link, leaving each release sharing a separate version of the file.

Figure 1 shows the relationships among a component hierarchy, releases, files, and versions of files in a single family. Note that releases, like files, are managed by a component.

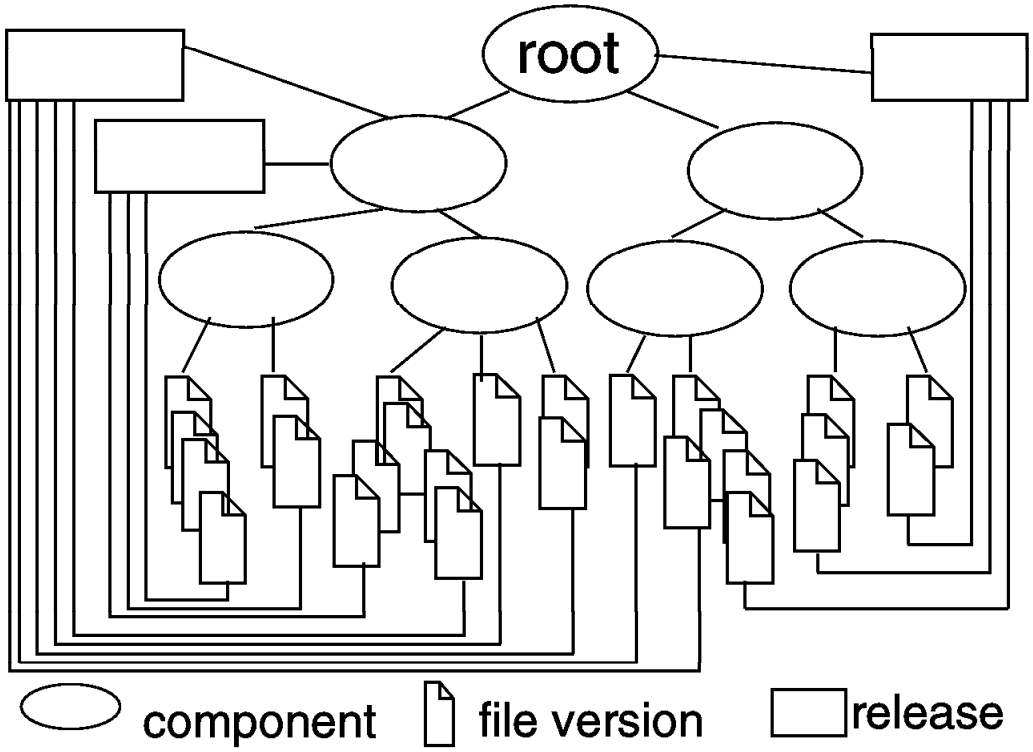


Figure 1. Relationships among Components, Files, File Versions, and Releases in a Single Family

1.5 Integrated Problem Tracking

CMVC supports monitoring and managing changes to software configuration items by integrating problem tracking with version control and release management functions.

1.5.1 Defect and Feature Processing

CMVC provides problem tracking for both feature and defect changes. Users submit requests for new features and reports of defects to CMVC. Defects and features must be associated with a particular CMVC component.

As you might expect, defects and features can be open, accepted, or rejected. Once accepted, defects and features are processed through various phases until they reach resolution and are closed. They may pass through design, size, and review states. Before closing, a defect or feature can be worked backward through these steps, under appropriate circumstances. On closing a defect, the original submitter can be requested to verify that the defect has been resolved or the feature has been implemented satisfactorily.

1.5.2 Track, Level, and Release Processing

CMVC supports the integration of problem tracking with release management by allowing the association of particular defects and features with particular releases. The CMVC track relates a set of file changes integrated into a release with the defect or feature that authorizes them.

CMVC also supports release management with the CMVC level. A level allows the easy management of a large release by breaking it into subsets according to functional, sequential, or other principles. Each level is defined by a set of tracks that are to be integrated into the release incrementally. All specific versions of the specific files associated with the tracks for a level are *extracted* into UNIX directory space so they can be integrated in a series of separate steps called *level members*. The required versions of all files implementing a release can be extracted according to several criteria, including by level.

Tracks and levels are processed in a configurable manner such that their status can be easily monitored as they progress through several phases. Tracks can go through optional approval, fix, level (integration), and test processing. Tracks and levels can be worked backward through the process under appropriate circumstances.

1.5.3 Change Control

Multiple sequential versions of a file can be controlled by CMVC. To ensure orderly changes in files, files are checked out before changes are made, and they are checked in afterwards. Once a file has been checked out, it cannot be checked out again by any user until the original user checks it back in. Check in and out can be associated with a given track (and therefore with a particular release, and defect or feature).

Any recorded version of a file can be extracted for reference purposes. The version specified can be either the current version, the version current at a given date, or the version associated with a particular version ID.

1.5.4 Audit Trail

CMVC ensures that an audit trail is maintained that explains, for every CMVC file, who modified the file, when it was changed, and why. CMVC provides traceability forward from the defect or feature to all releases in which it is resolved, and backward from the specific version of, or set of changes in a given file to the defect or feature that is related to the file.

1.5.5 Users and Host Lists

CMVC assigns all valid users a CMVC user ID. Through a host list the CMVC user ID is associated with a UNIX login ID and UNIX host. User IDs are recorded on a family basis. CMVC records and uses user data in association with many actions. Users, for example, own and manipulate all components, releases, defects, and other CMVC objects.

1.5.6 Components Control Access to Files and Releases

CMVC provides a mechanism to control user access to development data associated with components. All such access is through CMVC actions. All CMVC actions are grouped into subsets defined as *authority groups* A user can be assigned to one of more authority groups in a single component's *authority list*. Users implicitly acquire some access, can receive access explicitly, or receive access implicitly by inheritance from ancestor components access lists. Users can explicitly grant, or deny, access authority to users over components which they own.

CMVC provides for automatic notification of interested users of actions it takes against components, files, releases, and other CMVC objects. Notification is provided by electronic mail, so notification does not depend on a user invoking CMVC on a regular basis. Interest is configurable by component and type of action. Interest is defined by a user's membership in an interest group.

1.5.7 Configurability

CMVC provides configurability in the fields of records supporting features, defects, files, and users. CMVC also provides configurability in the processes that it enforces as objects (files, features, releases) move through the various states that it recognizes and controls. Finally, CMVC provides configurability in its actions, allowing the CMVC end user to program user exits, which will automatically execute under prescribed conditions in association with CMVC actions.

The reader who wants a more complete and detailed description of CMVC concepts, terminology, and functions is advised to study *Understanding IBM AIX Configuration Management Version Control/6000 Concepts*.

Chapter 2. Looking at CMVC through Customers' Eyes

In this chapter we describe how we researched this book and give a brief history of CMVC and its salient qualities. The chapter also summarizes the results of our research. The examples cited in this chapter are described in detail in Chapters 3, 4, and 5.

2.1 Data Gathering

We identified a small group of CMVC customers who were willing to help with our research and were representative in various ways of the larger set of CMVC customers. We visited several CMVC customer sites and exchanged telephone calls and electronic correspondence with other CMVC customers.

2.1.1 Customers Studied

We visited these external IBM CMVC customer locations:

- Intelligent Services Platform Division, MCI Telecommunications Corporation (MCI) in Colorado Springs, Colorado—A major U.S. and international long distance telephone service company
- SEIKO Communications Systems Inc. (SCS) in Portland Oregon—Provider of a radio paging service and pager receiver integrated into an electronic watch
- Continental AG (Continental) in Hannover, Germany–A large company owning several international tire and rubber manufacturing companies
- BT Development and Procurement, BT Laboratories, British Telecommunications PLC (British Telecom), Martlesham Heath and Ipswich, England–The largest provider of telecommunications services and telephone equipment in the United Kingdom.

We also visited two IBM divisions:

- The RISC System/6000 Division Development Laboratory (IBM Austin) in Austin, Texas—Developer of the AIX/6000 operating system, other software products for both AIX and OS/2, and the RISC System/6000 family of computers
- The Storage Systems Division's Development Laboratory (IBM Tucson) in Tucson, Arizona—Producer of workstation software and hard disk drives, disk controller subsystems, and tape storage products.

2.1.2 Interviews and Questionnaire

The authors interviewed managers, software configuration managers, developers, system administrators, and other technical end users at the customer sites in an attempt to get many perspectives on CMVC.

Interviews were guided by a questionnaire we developed which asked a series of questions to determine how and why customers used CMVC, and what they particularly valued about it. Different questions were asked of people playing different roles in the customer organization. Questions asked concern previous experiences with home-grown or other commercial CM tools, how CMVC is used presently, and which benefits are gained by its use in this manner.

We also talked with the developers of CMVC about its history, design, and implementation.

2.1.3 Demonstration and Examination

We watched demonstrations of locally developed front-end and back-end tools built around CMVC. We examined sophisticated management reports and graphical charts built by such tools that extracted and manipulated data from CMVC. We saw custom CMVC clients that had been developed for unsupported platforms. We watched how developers and end users interacted with CMVC and other development tools at their workstation. We examined customer use of CMVC server-end tailoring mechanisms, by studying the use of customer-developed user exits, configurable fields, configurable processes, and specialized input field choices lists.

2.1.4 End User Support Strategies

We looked over presentations used by CMVC customers to introduce CMVC within their organizations. These presentations, used to gain acceptance and approval of CMVC, stated how customers intended to use CMVC and which benefits they expected. We examined CMVC end-user documentation and education materials that customers had developed to assist new CMVC users install CMVC clients and begin using CMVC. We visited CMVC help-desk staff and inquired about ongoing end-user support procedures.

2.1.5 Computers and Networks

We toured computer laboratory facilities, examined network and remote access schemes, and noted the numbers and varieties of hardware and software platforms serving as CMVC clients and servers.

2.1.6 Lessons Learned

We solicited advice on how to successfully implement software configuration management using CMVC on projects of different sizes. We also asked for hints on how to implement CM with CMVC across multiple sites and a companywide standard. We listened as customers explained how their use of CMVC evolved as they became more familiar with its strengths and features.

2.2 Organizing the Data

Returning from these interviews, we organized and consolidated our notes. We concluded that discussing our findings on a customer-by-customer basis was not very informative. Instead, we decided to look at CMVC from the point of view of the degree to which customers use it and the size of the project or number of projects that customers manage with CMVC.

We divided customer use of CMVC into three broad groups for the purposes of this discussion:

- Very large-scale installations (IBM Austin, IBM Tucson)
- Medium-sized single projects or companywide standardization on CMVC for use on all projects (IBM Tucson, MCI)
- Small-scale or initial project use of CMVC (BT Laboratories, SCS, Continental).

We organized our findings by looking for both unique uses and common themes among customers who fall within each category.

2.3 Typical CMVC Customers

CMVC appeals to a broad variety of companies, a mixed group of businesses engaged in developing computer software for a variety of purposes. The companies are in a variety of industries and geographies. Many are part of large, or even multinational, entities. But, the decision to purchase and use CMVC is often made by a small company, or a single project or department within a larger entity. Occasionally the driving force behind the decision has division-level or corporatewide responsibility.

The amount of software or number of applications these companies develop covers a broad range, as do the importance and role of the software or applications in their business.

Thus, there is no typical CMVC customer profile. The customers whom visited are representative in many ways of the variety of businesses that are CMVC customers. One way of segmenting CMVC customers might be in terms of the type of business a company conducts, and the relationship between how a company makes money and the software it develops. Looked at in this way, a given customer may fall into several categories. In this section we group CMVC customers in this manner, pointing out some general characteristics of each category and describing how these impact CM tool requirements.

2.3.1 Software Vendors

For most CMVC customers, developing software is a means to another business end. For software vendors, however, software is their main product, contributing directly to the company' profitability. It is not surprising, therefore, that a tool providing automated software configuration management would be a mission critical necessity, not a convenience or nicety.

Software vendors use an automated CM tool to help them maintain a consistently high quality in their software. Such a tool can provide them with the management information necessary to manage and allocate their development resources to ensure the timely delivery of new software products and the efficient and ongoing maintenance of older software products. Software vendors also need to manage a significant amount of online and hard copy documentation in conjunction with software product releases.

Software vendors frequently use a variety of languages and development tools and target multiple, often nonhomogeneous platforms for the execution of their products. Software vendors sometimes require that new tools be integrated into the existing development environment and coexist with other tools. They require that the new tools lend themselves to being “programmed” to further this integration.

Software vendors frequently start small and grow rapidly, often in unexpected directions. Employing primarily computer programmers, they often attempt to build their own solutions to software configuration management problems when, as small companies, their requirements are fairly simple and their resources are thin. If they first look for a commercial tool to automate CM, they often conclude that they have neither the budget nor the requirement for a comprehensive product.

As the software vendors grow, so grows their dependence on their CM tools. If they have taken on the development and maintenance of their critical CM tool, they soon discover that it takes on a life of its own quite apart from the development

efforts it is supporting. The maintenance effort often consumes increasing resources, while the tool itself becomes more fragile and less reliable.

Requirements for development tools that are to be built locally are rarely analyzed thoroughly, and the tools themselves are not designed for generalized use. Often these tools live on from one project to the next, taking on new features, functions, and project-specific characteristics. Moreover, tools built for internal consumption are rarely implemented with the rigorous software engineering methodology used for the product software.

The further along this path a software vendor goes, the more difficulty it finds in replacing its home-grown CM tool with a commercial product. The more adapted its CM tool becomes to its unique development environment and process, the more convinced a software vendor becomes that its CM requirements are unique. Ironically, a vendor often concludes that the same full-function product that was too complex originally now fails to provide as much function as its custom solution provides. The available commercial products may seem too restrictive or inflexible to implement the unique processes that the vendor has evolved.

IBM Austin is in many ways representative of the software vendor industry, because it develops and maintains operating system, networking, and workstation application software for AIX and OS/2, as well as the RISC System/6000 family of computers, peripherals, and Xstations. Some of its products target other vendor's UNIX platforms as well.

IBM Tucson is also representative in that it develops and maintains a stand-alone software product called ADSTAR Distributed Storage Management (ADSM). This product provides backup, restore, archive, and retrieve functions across a wide variety of heterogeneous computer platforms.

2.3.2 Hardware Vendors

The software that hardware vendors develop forms a significant and growing component of their main products. Most of it is microcode, embedded within their hardware. Automated tools developed by hardware vendors may be critical in automating various aspects of their engineering and manufacturing processes.

The fact that software configurations should be controlled and managed, at least as rigorously as hardware configuration data, is self-evident to hardware vendors. Written in various languages, including assembly languages, hardware vendor software is often either cross-compiled on a development workstation or downloaded for assembly on target microprocessors.

Hardware vendors may want to dovetail configuration management practices in their hardware and software development processes. At a minimum, they will need to coordinate schedules between hardware and software development efforts. To control software problem reporting and tracking, deliverable media archive or engineering documentation, and release management, a hardware vendor is likely to start out using CM tools designed for hardware development and maintenance. As the software component grows and becomes more complex, the two CM worlds diverge, and it becomes obvious that tools with greater affinity to the development platforms and tools tailored for the software development process are necessary.

IBM Austin and IBM Tucson are both representative of hardware vendors. Both develop and maintain hardware, peripherals, and/or subsystems with significant

software content that they control with CMVC. SCS also represents this type of customer. Microcode developed in this Portland subsidiary is embedded within microchip components of watches that the parent company manufactures in the Orient.

2.3.3 Manufacturing Companies

Companies whose main business is manufacturing are also heavily dependent on software that they develop themselves. Originally, these companies may only have developed business applications for information management. They now find themselves increasingly dependent on software critical to both their manufacturing and product engineering processes.

Typical manufacturing companies will have heritage applications and computing facilities that predate the open systems explosion of recent years. As a result, these companies often target their applications for execution on proprietary mainframe and midrange hosts, but they also are doing more and more software development on PC desktop computers networked together with the hosts.

Manufacturing companies have not been ignoring the open systems revolution, however. They also find themselves using UNIX workstations as development platforms and even targeting a few applications for multiple vendor UNIX platforms. They are not as likely as a software vendor to have a wide variety of vendors, UNIX or otherwise, in their development shops, though.

These companies are less likely to have developed their own CM tools and are less heavily invested in development tools or methodologies than software and hardware vendors. They may have been using CM tools on their proprietary hosts, but not to the maximum extent they might. Problem tracking, for instance, may not be as fully institutionalized as version control. However, if they have significant exposure to the hardware design and manufacturing automation revolution of recent years, they may be more readily convinced of the productivity gains to be had from tools that enhance the development environment for the programmer and coordinate the efforts of the programming team.

As they move to UNIX-based computers and rely increasingly on PC-based application development environments, manufacturing companies are seriously evaluating 4GLs and other modern tools to support and automate software development on the LAN-based platforms. These companies recognize that their heritage software will continue to coexist with their newer workstation-based applications, but that their host-based CM tools will not suffice. They are, therefore, looking to LAN-based CM tools to service both large and small, proprietary and open, platforms.

Continental is representative of a manufacturing company. It has a staff of 400 maintaining and developing business, manufacturing, and engineering applications relevant to the design, chemistry, and manufacture of automotive tires.

2.3.4 Services Providers

Many companies that are neither computer hardware nor software vendors find themselves developing a lot of software these days. These service providers develop and maintain large and numerous application programs that are critical to providing the services that are their primary products. To these companies,

software engineering plays its role in the context of engineering large and complex systems that have significant hardware, software, and human components.

These service providers, also develop both business and utility software for internal consumption. Because company workers become dependent on such software, its reliability and maintainability are critical to business success. These companies are likely to invest in tools to support their software engineering methodology and technology, including CM tools. These companies are likely to be using C, C++, as well as 4GL products.

Service providers typically develop a large percentage of their software for UNIX-based workstation and server-sized computer platforms available from a variety of hardware vendors. They may use PC-class workstations as development platforms, too. Some may still have applications executing on proprietary midrange computers also. These companies may want to control and manage all of their software with a single CM tool. They generally look to open systems vendors for solutions that are compatible with that environment and execute on a variety of hardware platforms, including non-UNIX desktops.

MCI and BT Laboratories are representative of service providers. Businesses such as telecommunications companies may sell hardware, or even software, but their mainline business is the provision of telephone and telecommunications services. SCS also fits into this category. Although the paging hardware (built into watches) is a product, the network of transmitters, receivers, and switches that they also develop provides the backbone of its radio-based services.

2.4 Scalability of CMVC

We found that CMVC scales tremendously, in terms of the volume of files, volume of metadata, numbers of users, numbers of families, components, tracks, releases, and other CMVC objects.

2.4.1 Very Large Scale Application

On the high end is IBM Austin, where CMVC supports more than 3000 users. The vast majority of these users work on one very large project, developing and maintaining multiple production releases of a single operating system's source code. IBM Austin, has one family containing 1800 components that manages their access to over 600,000 files. The users at this installation access CMVC around the clock, literally from around the world, and have done so with virtually no downtime attributable to CMVC since its inception. Seven large-capacity server computers in Austin support hundreds of LAN-attached PCs, Xstations, and RISC System/6000 workstations as well as numerous remote locations connected by WAN and dial-up mechanisms.

2.4.2 Small Scale Application

On the low end, CMVC is successfully used at SCS to support just 15 users working on a single project. Here, only 20 users are working on two projects whose data is contained within a single CMVC family. Thirty components are used to manage about 3000 files. Here, also, the hardware support is much smaller. One RISC System/6000 serves as a server, and it and one other RISC System/6000 workstation, support all users on a combination of Xstations and PCs.

2.5 Limited Functional Use of CMVC

Our study showed that CMVC is sometimes used in a limited manner, as, for example, where CMVC must integrate with and augment existing CM tools, or where CMVC is being introduced to a new project, but full configuration management functionality is not yet required. The two scenarios we encountered most often were use of:

- Version and release control only
- Problem tracking only.

2.5.1 Version and Release Management Only

When a company is just introducing CMVC, or at the early stages in software development, it may prefer to use only the version and release control mechanisms in CMVC. Continental, for instance, is not using problem tracking yet on its three projects, because none of them has gone into production use.

At IBM Tucson, however, a VM-based tool has long been used to track problems, and not all problems that arise will necessarily be solved by changes to software under CMVC control. Some projects therefore keep problem tracking on the mainframe and use CMVC primarily to manage the software.

2.5.2 Problem Tracking Only

Although a customer may not purchase CMVC to use it only for problem tracking, once it is in house, specific cases where only this feature is required can appear. IBM Austin, for instance, had an established practice of using INTERLEAF for version control of individual document source files, before CMVC was developed. It did not abandon use of INTERLEAF for version control when CMVC became available, but it did begin to use CMVC for problem tracking.

One reason was that IBM Austin needed to integrate control of product code and documentation whenever a new release was about to be put into production. To achieve this it creates components representing the completed release-level documents. Using a problem report in conjunction with the new software release, it integrates the binary images, of both the printable documents and the online information libraries, with the software build for the formal release. It brings these document images under control of CMVC managed by the documentation components.

IBM Austin also needs a mechanism by which defects reported by users can be assigned either to documentation or code or both, or reassigned to documentation or code, for resolution. To do this, IBM Austin needs components representing the documentation in the same family as those components by which the source code is controlled. These defects are registered against the same documentation components used to manage the binary images.

2.6 Standardizing on a Single CM Tool

CMVC customer experience indicated that CMVC lends itself to becoming the companywide standard CM tool, whether by executive *fiat* emanating from the top of an organization, or democratic consensus, whereby individual projects increasingly choose to use CMVC because of its strong internal reputation.

2.6.1 Development of Formal Standards

Continental selected CMVC after a long and thorough trade study of alternative competing solutions. Use of CMVC is being implemented in a very rigorous, carefully planned, incremental manner across the company's several development sites. Continental developed specific standards defining how CMVC is to be used and manuals focused on specific end-user roles and prepared and formally presented an introduction to CMVC concepts and plans to various internal audiences.

2.6.2 Evolution of De Facto Standards

At IBM Tucson, and to some extent IBM Austin, the favorable reputation of CMVC among AIX developers spread at the grass-roots level causing demand for CMVC by various development organizations. Informal end-user support was initially provided by a bulletin board service staffed at the original CMVC development lab. Demand for end-user training led to the development of several end-user CMVC training courses. The experiences of previous projects and the common training received by new CMVC users became the basis of the development of informal *de facto* standards on how to use of CMVC.

Formal internal recognition of the benefits of CMVC led to the declaration of CMVC as the CM tool of choice for all new development projects at IBM Tucson. An organization whose responsibility includes developing and maintaining automated tools in support of hardware and software engineering now has the responsibility to support rollout of CMVC on new development efforts.

2.7 Importance of ISO to CMVC Customers

A very interesting finding is that obtaining ISO certification is a very important goal to most CMVC customers studied. ISO certification is critical to CMVC customers who are in the business of selling computer hardware and software on a worldwide basis. IBM Austin, IBM Tucson, and IBM Toronto each sought and acquired ISO certification. Such certification is also essential to companies selling services, if they sell to international markets. Therefore CMVC customers such as MCI and BT Laboratories also sought ISO certification. One customer even admitted to losing contract bids because it lacked ISO certification and noted ruefully that its competitors already possessed it.

All CMVC customers studied perceive CMVC as supportive of their efforts in this area to one degree or another. Many are currently exploring additional ways that CMVC can aid in their attainment or retention of ISO certification. Two main themes were observed repeatedly:

- By automating CM procedures, CMVC contributes to the establishment of a well-defined, orderly, and repeatable development process, which itself implements various aspects of the ISO specification.
- Some customers use CMVC to control the very procedures documents that define the development process. These same customers use CMVC problem tracking to identify and resolve deficiencies in those procedures and control changes to the documents that define them.

Those customers who focused on ISO certification observed that use of CMVC generally improves their management of the software development process. BT Laboratories, for instance, believed it was no coincidence that it failed an internal

audit before implementing CMVC and passed it afterward. IBM Toronto began to use CMVC when it sought ISO certification to manage, record, and track changes in its procedure manuals. It also uses CMVC to manage and track deficiencies identified during internal and formal ISO audits; IBM Austin set up a separate family primarily for this purpose.

2.8 Platform Choices

The number of operating system and hardware platforms supported by CMVC clients and servers has grown since its initial offering. This evolution has been in specific response to customer needs.

2.8.1 Locally Developed Clients

Customers requiring a specific platform-dependent client are generally happy to serve as *beta* test sites for these new clients. MCI, for instance, provided early customer evaluation of the CMVC Version 2.1 GUI client and the OS/2 CMVC client. Continental volunteered to help with the customer evaluation phase of the CMVC client for Microsoft Windows. Generally, at the time of this study, customer needs were met by the support currently available on Sun, HP, Windows, OS/2, and AIX.

Some customers, notably IBM Austin and IBM Tucson, have additional requirements and sufficient resources to justify their own implementation of additional clients, although IBM Toronto does not perceive sufficient market demand for such clients as part of the CMVC product.

2.8.2 Developing on and Targeted to Multiple Heterogeneous Platforms

A significant finding is that CMVC is considered appropriate not only for controlling software developed on and targeted to UNIX platforms but also for other heterogeneous hardware and operating system platforms. In fact, CMVC customers choose CMVC/6000 on a RISC System/6000 as a CM solution even in environments where AIX was not used for production at all, and where AIX development played a very minor role.

Customers use CMVC to control software targeted for IBM's VM, MVS, and OS/2, as well as for Microsoft's DOS and Windows. CMVC customers control code destined to run on DEC's VAX/VMS, Sun's SunOS and Solaris, Hewlett-Packard's HP-UX, and many other platforms. IBM Tucson uses CMVC to control source for ADSM clients that run under 13 different operating systems on seven different hardware platforms. SCS and IBM Tucson also target multiple microprocessor environments that might have no commercial operating system.

2.9 Applicability of CMVC

CMVC customers find CMVC appropriate for controlling a wide variety of data, including but not limited to source code. CMVC customers use CMVC to support development efforts using a wide variety of language technologies. CMVC is used to support all types of software, as well.

2.9.1 Data and Files

All customers use CMVC to control source code and operating system command language scripts. Additionally, they use CMVC control product documentation, departmental procedures, design specifications, charts and drawings, test cases and results, and various binary file formats, such as electronic publishing internal format document files, online product information images, and application executables. Automated build instructions and make files are also candidates for CMVC control.

In addition to controlling source code IBM Austin uses CMVC to control the InfoExplorer information libraries that contain the online versions of AIX and related product manuals and the COBOL object code that it receives from software vendors. MCI and IBM Austin control publication formatted files produced by INTERLEAF. IBM Tucson information developers control BookMaster source under CMVC.

2.9.2 Language and Technology

CMVC is an appropriate tool for controlling any source code language, from assembler to 4GL, to interpretive languages. It is useful for controlling intermediate formats from which code generators produce source code as well.

Typical uses include:

- IBM Austin and IBM Tucson use CMVC to control C source code and assembler.
- BT Laboratories uses CMVC to control C++ and C.
- Continental uses CMVC to control files written in APT (a 4GL for Sybase) and C.

IBM Tucson also uses CMVC to control files generated by CADRE TeamWork from which SmallTalk code is later generated. Command language scripts in languages such as REXX, C shell, and Korn shell are managed under CMVC at all customer sites.

2.9.3 Types of Software

Configuration management is critical, whether the application is a product in the hands of paying customers or merely used within the company by its own people to do their work. Whether the application serves business purposes such as providing management with business data, storing a bill of materials or managing inventory, the application requires configuration management. Tools developed to enhance productivity in software engineering, support design modeling, or perform chemical analysis require careful control if business processes depend on them. The customer sites studied use CMVC to control operating system, application, embedded or distributed, software in all categories.

2.10 Customizing CMVC

CMVC is shipped with quite a lot of customization already done for the customer. The CMVC customer experience documented in this book shows generally that the smaller the project, or the newer the customer is to CMVC, the more appropriate this default customization is. CMVC's potential for further customer-specific customization is, however, also widely realized.

Larger projects by their nature are more complicated and make good use of the customization features of CMVC. The IBM Austin experience bears this out. Sites where multiple projects use CMVC find a greater variety of requirements and therefore have greater likelihood to need further customization. IBM Tucson typifies this customer situation. Sites that have used CMVC for long periods of time evolve new requirements, which they solve using CMVC's configurable features. MCI shows this clearly.

The most significant ways in which the customers studied enhanced CMVC are:

- Configurable fields
- Choices lists
- User exits
- Authority and interest groups
- Configurable processes.

2.10.1 Configurable Fields

Configurable fields are customer-defined fields added to the fixed record structure for certain CMVC object types. Command-line and graphical user interfaces and reports are automatically adjusted to accommodate these additional fields. Configurable fields provide a mechanism for the customer to store and retrieve data relevant to defects, features, users, and files within the database, as if it were CMVC metadata. Configurable fields are in use or planned at all customer sites studied.

A customer using CMVC in a simple and straightforward manner may discover the need for few, if any, configurable fields. For example, Continental has none, and SCS has only one. A customer using CMVC on a larger scale or for multiple projects soon discovers many potential uses for configurable fields. MCI uses a combination of configurable fields and user exits to ensure that file names are formed properly when files are extracted to NFS file systems and mounted from non-UNIX operating system hosts.

IBM Austin still uses CMVC Version 1, which does not offer configurable fields, but has many instances where one is necessary. Instead, they use extension of choices lists for a fixed field as a way of overloading a single field to combine the data that otherwise would be stored in multiple fields.

2.10.2 Choices Lists

A choices list is the fixed set of values that can be applied to a given field in a CMVC object type. CMVC allows customers to modify the list of choices associated with several fields in several records. A field containing defect priority, for instance, would have one of several values, but the exact range of values and choice of words to express them will vary greatly from customer to customer, or even between two projects at the same site. Some customers modify choices lists for a single field over time, as the project moves through its development phases. Others modify the list with a set of values that is valid for all time.

The simpler the customer's use of CMVC, the smaller the reliance on choices lists customization. Continental does not use choices list, whereas SCS uses them for only one CMVC object type. IBM Austin uses choices list heavily. It modifies the choices lists for certain fields in a defect, when the operating system release nears a release date. It also makes a pass through all open defects, replacing the old values with values from the new list.

2.10.3 User Exits

User exits are a particularly valuable type of customization. User exits allow the customer to extend CMVC functions, causing additional actions to be taken before or after the CMVC command executes at the server. User exits are shell or executable programs. They have access to the metadata of the record being manipulated, as well as the parameter data being passed to the CMVC command.

A customer who uses CMVC in a very simple manner, again, does not need to create user exits. Continental uses none; SCS has but one planned. Numerous and various uses, however, are found with most customers. The more complex the configuration management situation, the more likely the customer is to implement user exits. Medium-sized customer sites, such as BT Laboratories and MCI, have defined a few user exits. IBM Austin needs many more user exits than it can implement because it had not upgraded to CMVC Version 2 at the time of writing.

2.10.4 Authority and Interest Groups

CMVC defines the set of users authorized to perform CMVC actions and the set of users to be notified by electronic mail whenever CMVC actions occur. CMVC customers can define additional authority and interest groups and associate individual CMVC users with these groups. Examples of this customization are rarely found, indicating that the groups defined by CMVC are sufficient for most customers. Customers are not always sure how to use these features in a sophisticated way, and smaller customers tend to maximize access and notification. SCS, for instance, put all team members in the same authority and interest groups and used group definitions shipped with CMVC.

2.10.5 Configurable Processes

CMVC Version 1 offered essentially two choices in process: binding and non-binding control. CMVC Version 2 enables configurable processes. Configurable processes pertain to two areas of CMVC:

- A component process choice indicates which combination of design-size-review (DSR) and verify subprocesses are to be applied to defects and features associated with that component.
- A release process choice indicates whether CMVC track subprocessing will occur on that release, and which combination of level, approval, test, and fix subprocesses is required for the tracks.

CMVC is shipped with a predefined set of release and component process choices. Generally, customers are content with the process choices offered, but CMVC allows customers to define new labels for a given choice or to define a new choice that represents a unique combination of subprocesses not predefined. These choices are used when components or releases are created, but processes can be changed after the fact under certain conditions.

Generally, customers experiment with more rich subprocess combinations only after they are experienced with CMVC. A few combinations of subprocesses therefore are popular with most CMVC customers. Most prefer to use the DSR subprocess only with CMVC features. Use of the verification subprocess is not appropriate in environments where the person who files the defect is likely to be the same person who fixes it.

MCI and IBM Tucson, like most customers, use track processing, but often in a limited manner. MCI, for example, does not use the verification subprocess and uses the test subprocess only in limited cases. Continental does not use problem tracking at all and therefore does not use component subprocesses. At IBM Tucson, where many independent projects are using CMVC, the choice of subprocess varies greatly between projects. At IBM Austin, where CMVC Version 2 is not yet in widespread use, binding control is the norm.

2.11 Extensions to CMVC

CMVC extensions developed and maintained by CMVC customers were found at customer sites where CMVC was used on more intricate projects, or in a more complex development environment. Extensions include what we would characterize as:

- Back-end tools
- Front-end tools
- CMVC clients
- Report generation.

As a customer becomes heavily dependent on CMVC, or as the sheer volume of configuration management data grows, the customer needs more sophisticated ways to enter, extract, and evaluate CMVC data. At the same time, the customer can justify additional programming tasks to make that data available to a wider audience than just the development team.

2.11.1 Back-End Tools

A common type of extension is the automation of local and remote build processing. CMVC does not provide explicit mechanisms to support build automation. Build processing is often linked to the CMVC release and level extract commands, however. Every customer site visited had created some type of build automation and linked it to CMVC.

After build automation, we found that a customer is most likely to extend CMVC with a tool to extract and manipulate metadata from CMVC. This metadata is retrieved by using CMVC report commands. It is then massaged, formatted, evaluated, and turned into report or graphical format. Most customers gather information, which they use to track and plan the development schedule. They focus particularly on problem tracking and release management data. This tendency is more pronounced at the larger CMVC installations. IBM Austin provided the most intricate examples, followed closely by IBM Tucson.

2.11.2 Front-End Tools

A less common type of extension to CMVC is a tool that automatically inputs data to CMVC. This type of tool is developed where CMVC must coexist with other CM tools.

The need for a front-end extension to CMVC is more pronounced at large CMVC installations, such as IBM Austin and IBM Tucson. One such example can be found in a VM-based customer support program. When a customer problem is

determined to be a defect in AIX, the VM-based customer support program automatically opens a CMVC defect. A similar tool is found at IBM Tucson.

2.11.3 Custom Clients

In environments where CMVC does not support a client for a specific platform, but the customer is heavily dependent on that platform, customer-developed clients are found. Nonproprietary vendor platforms support convenient remote access to a workstation or server on the LAN on which a CMVC client is installed. IBM Austin and IBM Tucson, because of their predictable dependence on OS/2 and VM/CMS, abound with extensions of this sort.

Custom clients are also a response to the need for a very limited and well-defined user interaction with CMVC. When a customer has a large staff interacting with CMVC and very clearly differentiated end-user roles, the requirement for custom clients, or mini-applications that act like a CMVC client, arise.

One tool of this sort is used at IBM Tucson by personnel who deal with product defects originating at IBM customers. This tool executes on VM, where it manipulates the VM-based RETAIN database. If a problem relates to software, this tool issues the CMVC commands to create and manipulate CMVC defects on behalf of the user.

Another tool of this sort is found at IBM Tucson. It is an AIX-based, locally developed comprehensive edit-compile-debug tool. It provides access to all CMVC functions that programmers need.

One other front-end extension is the modification of a commercial development environment product to allow end-user access to CMVC indirectly. This is very convenient to the user who primarily uses one tool or development environment. This solution is most likely to appeal to a company that is not in the habit of custom coding its own support tools.

The WorkBench/6000 Development Manager menus can be modified to perform CMVC check-in, check-out, release, or file extract actions, for example. Some developers at Continental use WorkBench, and at BT Laboratories they use HP's SoftBench on one project.

2.11.4 Report Generation

CMVC customers become more dependent on reports generated by CMVC and from CMVC metadata as time goes by and they become more familiar with CMVC capabilities. Report generation has been raised to a fine art at IBM Austin, which has had the longest exposure to CMVC. Standard reports generated by CMVC suffice for needs at Continental and SCS presently. Use of these reports clusters around two purposes:

- Managing development schedules and resources
- Collecting and reviewing software quality metrics.

The latter purpose is fairly rare, although all customers are very interested in exploiting it.

2.12 Common Reasons for Choosing CMVC

The customers visited—large, medium, and small —cite several CMVC qualities that they value highly, whether they barely use them yet or already stretch them to their maximum. These qualities are:

- Reliance on open systems standards
- Flexibility, tailorability, extendibility
- Comprehensive process model
- Scalability
- Availability of appropriate CMVC client and server platforms
- Methodology and language technology independence
- File format and file content independence
- Granularity (independent usability of distinct functions)
- Enablement of specialized add-on tools and extensions
- Ability to integrate into existing development environments.

Another quality appreciated by these customers is the fact that there is choice in the underlying relational database and version control products used by CMVC.

2.13 Benefits of Using CMVC

The benefits of using CMVC are best expressed in the words of the CMVC customers interviewed during this study. When asked what are the most important benefits of using CMVC, a BT Laboratories manager replied “CMVC provides traceability in the development process. It gives us a better feeling that we have put the right pieces together.” An MCI manager responded that “Our software process is more predictable and overall we have a better process.”

When asked what has changed since they began using CMVC, an MCI manager replied, “Our developers are more quality oriented.... They have a more rigorous development approach... [and] we have experienced fewer slippages during testing.” Asked how CMVC was accepted, the same MCI manager responded, “Managers and the metrics people loved it immediately.”

Although none of the customers has done a systematic cost-benefit analysis of using CMVC versus using no automated CM support, each discounts the possibility of doing without CMVC in the future. At Continental, we were told that “CMVC is an absolute necessity!” At MCI, CMVC is an “absolute necessity for mission critical operations!...CMVC definitely increases product quality...[CMVC] produces a more predictable level of quality of software... [and] we need to revisit Software less often!” A BT Laboratories manager considers CMVC both “flexible and comprehensive...offering the granularity you need as the project evolves.”

Chapter 3. Use of CMVC on a Small Scale or in the Initial Phases of Development

This chapter looks at the characteristics of CMVC usage by customers who are using CMVC to support a small-scale development effort, or the initial phases of a larger development effort, or the first of many development and maintenance efforts that will transition to CMVC. This chapter examines which issues are most significant to this group of CMVC customers. CMVC customers who want to evaluate CMVC by using it first on a small project will experience issues similar to those discussed in this chapter.

3.1 Representative Customers

For this study we visited three representative customer sites. The companies do not have a lot in common, in terms of the types of business they are in and the role software plays in their business. Although none of these companies is a software vendor, their application development activity is integral to engineering or developing the products and services that form the basis of their businesses. These companies are briefly described below.

3.1.1 SCS

SEIKO** Communications Systems, Inc., sometimes known as SCS, was formed by SEIKO EPSON Corporation and HATTORI SEIKO Corporation, Japanese manufacturer of precision timepieces and other consumer electronics. SCS is located in Beaverton, a suburb of Portland, Oregon. The company employs about 20 people who develop software. Another 40 people provide administrative, marketing, network operations, and business support for its products and services. SCS was formerly an independent company known as AT0.

SCS's main products are the SEIKO RECEPTOR** Messagewatch** and the SEIKO RECEPTOR Message Delivery System. The electronic Messagewatch incorporates a radio pager receiver that accepts messages from the message delivery system. The message delivery system consists of a central message routing subsystem and multiple widely placed transmission subsystems. At the time we met with SCS, the SEIKO RECEPTOR Messagewatch and paging service was available in both Portland, Oregon, and Seatlle, Washington; a demonstration service was installed in Paris, France. SCS expressed plans to expand the subscription service starting in the summer of 1994 with service in the region of Los Angeles, California.

The development project that uses CMVC is a 20-person effort in the preproduction phase of development. This includes the rehosting of the MDS onto IBM RISC System/6000s and PS/2s. In addition, other software under development executes in embedded microprocessors within the wristwatches. The software development environment consists of an IBM RISC System/6000 Model 580 acting as the CMVC and file server with two desktop models in the 3XX series hosting CMVC clients. These and other graphical desktops connected through a LAN allow access to the CMVC GUI client.

As might be expected, SCS emphasizes CMVC usage for source code and release control during the development phase. However, it anticipates that its CM

requirements will grow during the maintenance phase, when it will need to track which hardware and software components have been installed at which service sites in which geographic locations.

3.1.2 Continental

Continental is a large holding company owning about 50 medium and smaller companies in the rubber and tire manufacturing industry. The larger companies owned by Continental are Continental Tire and Uniroyal tire companies in Germany, Semperit in Austria, and General Tire in the United States. Continental's most important products are tires; it is the second largest manufacturer in Europe and the fourth largest in the world. From its headquarters in Hannover, Germany, Continental manages several manufacturing sites.

Application development at Continental is split across four different locations employing a total of about 400 people. Presently Continental does not use CMVC for the bulk of this development effort. It plans to make CMVC a company standard CM tool, extending CMVC support to a large percentage of its in-house development over time. Initially, Continental chose to use CMVC on development efforts that use the IBM RISC System/6000 as a development platform. Its plan is to use CMVC next on projects that use other UNIX-based development platforms.

The current efforts using CMVC on RISC System/6000s are developing applications targeted to execute under DEC** VAX/VMS** and Microsoft DOS on non-IBM hardware platforms. Each development group involves about 5 people.

The main CMVC emphasis for Continental is to enable proper management of the entire application development process. At present these projects are in the early stages of the development. But, when the applications are installed in the production environment, maintenance issues will be of great importance. CMVC's unique capability to integrate change management, configuration management, and problem tracking on heterogeneous platforms is of great value to this customer.

3.1.3 BT Laboratories

British Telecom is the biggest provider of telecommunications services in the United Kingdom. It has several development sites in England. The sites that we studied are the BT Laboratories at Martlesham Heath and Ipswich. CMVC has been in use for some time at BT Laboratories. After the first project successfully used CMVC, news of it spread by word of mouth, and other development projects at BT Laboratories began to use CMVC.

Groups at one site are currently developing speech applications in a heterogeneous environment. The development environment there consists of 15 Hewlett-Packard** and 2 Sun** workstations with CMVC clients integrated in SoftBench**, and the CMVC client and server executing on a RISC System/6000. The target platform is a Sun workstation.

At the other site, one group is currently porting an automatic test system for private circuits, originally written for SUN workstations, to IBM RISC System/6000 workstations. Once this application is completed, it will be installed at different sites and maintained using CMVC.

At these two BT Laboratories locations, CMVC is helping to manage a very large amount of rapidly changing source code under development by several major software projects. Each development effort involves about 15 developers.

The functionality of CMVC helps with release and change management but throughout the Laboratories, the main emphasis is on maintaining greater software quality. From BT Laboratories point of view, the availability of CMVC on different UNIX platforms makes CMVC well-suited to support large scale open system development.

3.2 CMVC for Small-Scale Efforts

The most immediate question that comes to mind when one sees a small-scale development effort using an industrial-strength, full-featured CM product like CMVC is, Why did the customer choose to make the investment?

The experience of the three customers studied indicates three common reasons:

- Small projects face many of the same CM challenges that larger efforts face. The fact that fewer people are programming does not imply that they will produce software applications that are any less complex or significant to the company.
- At the early stages of development these companies also are taking care to plan for growth in the size of the application and changes in the nature of their CM requirements.
- Automated CM is also seen as a productivity tool, which might be of even more impact to a small project than to a large project. Customers in this category typically do not use all of CMVC's capabilities immediately, and they usually do not implement CM as rigorously as they expect to later. CMVC meets the immediate needs of these customers and positions them for their projects' future.

3.2.1 Smaller Projects Need CM

Configuration and change management systems are put in place on small projects to solve and prevent the very same problems encountered on large efforts. These customers implement CMVC to enable them to:

- Control the source code changes and maintain development history
- Ensure re-creatability of previous versions during later test or maintenance activities
- Prevent the building of unreliable or incomplete application versions due to errors in component lists or incompatible combinations of software components
- Eliminate confusion on the exact requirements of maintenance tasks
- Track the status of maintenance and development activities.

3.2.2 Preparing for the Maintenance Phase

While these customers' applications are not yet in the maintenance phase of their lifetime, they are very conscious of the demands that this phase will introduce. Confident of the successes of their applications, these companies anticipate the requirements for integrated problem tracking with release management and version control once they have multiple production baselines to support. They want their developers and system administrators to be comfortable with their CM tools before those needs are evident. They want their configuration managers, build or release,

and test engineers to have had time to understand how to utilize their automated CM tools in a sophisticated manner before they reach this stage in their development effort.

They also seem to recognize unanimously the importance of a smooth software maintenance process to ensuring customer satisfaction with their products and services.

In short, these customers, while working on small-scale efforts, are not amateurs in the application development world. They do not translate “small” to “simple,” and they recognize that certain strategies necessary for success in software development fit projects of all sizes.

3.2.3 Increasing Developer Productivity

Not surprisingly, small projects have fewer people to devote to manual and clerical activities often associated with software configuration management. If 1 person in 10 or 15 needs to spend a significant amount of time performing CM tasks, instead of developing the new application, that significantly impacts average individual productivity. The same is true if one tester spends a day unraveling a configuration management error induced by sloppy book-keeping procedures. Perhaps because of its association with a Japanese company that views employment as a lifetime commitment with significant overhead, SCS voiced a firm conviction that investing in tools that increase programmer productivity is preferable to increasing the team size to achieve the same purpose.

3.3 Choosing a New CM Tool

Each of these customers had previous experience with either commercial or home-grown CM tools. In some cases, they were already using an unsatisfactory CM tool or product at the time they chose CMVC. The chief disadvantages of their previous CM solutions were primarily:

- Severe performance problems
- Functional limitations
- Lack of vendor support or excessive cost of internal support.

These customers took the time to analyze their CM requirements carefully. They also surveyed the available vendor products for software configuration management. They sought the best product available to match their current and anticipated requirements. Each of these customers independently concluded that CMVC is a solid commercial product offering good performance, unique functionality, and reliable support.

3.3.1 CM Tool Selection Criteria

In their search for a commercial CM tool, these companies created a list of decision criteria against which to measure potential CM solutions. Table 1 and Table 2 summarize the decision criteria that Continental used during its evaluation of CM tools. The weighting factors included in these tables are valid numbers for Continental, but they would vary from organization to organization.

Table 1. Decision Matrix: Functional Criteria	
Functional Criteria	**Relative Importance (1=low, 10=high)**
Version control	10
Release management	10
Change management	7
Product design	6
Configuration build	8
Role management	5
Report facility	5

Table 2. Decision Matrix: Nonfunctional Criteria		
Nonfunctional Criteria	**Description**	**Relative Importance (1..10)**
Price/performance ratio	Price (including primary and secondary costs) relative to performance	7
Technology, strategy, portability	Level of possible integration into present and future development activities; support for customization; availability on multiple platforms	7
Support, hotline, education	Support and assistance during introduction phase and during operation	8
Reference	Customer base	7
User interface, online help	Ease of use of the product	6
Setup, administration	Effort required for the introduction and administration	7

All of these customers pointed out to us that CMVC is unique for integrating configuration management and change control with problem and feature tracking. They observed that CMVC is available on the range of UNIX platforms that they typically use and supports appropriate PC-based clients.

3.3.2 Tool Support and Maintenance

It is not surprising that availability of vendor support and maintenance is included in the list of criteria as an important factor in the selection of a new CM product. This results from customer experiences with both vendor and locally developed CM tools.

Before installing CMVC, BT Laboratories had used a home-grown tool that had become functionally inadequate. Because it was difficult to maintain, enhancements and corrective tasks had become very rare, causing a significant drop in end-user satisfaction and productivity. Moving to CMVC offered sound functionality coupled with robustness.

SCS used a vendor product that over time failed to keep up with its needs in several ways. First, usage outgrew its performance capabilities. Gradually, SCS's requirements grew beyond the product's functional capabilities. Finally, the vendor

withdrew support for the product. Attempting to maintain and administer the product consumed increasing developer resources until it became clear that investment in a new CM tool was inevitable. After installing CMVC, SCS found that its CM tool administration and related system management support dropped from four people to one.

3.3.3 CM System Administration

All these customer sites report their system administrator is able to automate many of the standard administrative tasks directly related to CM. CMVC's command line interface is well-suited for use with **shell** scripts, **cron** jobs, etc.

CMVC is built upon a commercial relational data base product. Customers can choose the RDBMS with which they are most familiar to take advantage of the skills and procedures they already have in place for system administration tasks such as backups, restore, etc. Being able to leverage existing skills is very important to small projects which have fewer people to devote to CM.

3.4 CMVC and the Application Development Process

Having chosen CMVC as a new CM tool, these customers next explored how best to integrate CMVC into their software development process. Most small-scale users studied find CMVC flexible enough to fit into their current development process without creating unnecessary overhead. CMVC flexibility also offers these customers the possibility to grow and change the CMVC process to accommodate their future development process, no matter how complex it might become.

3.4.1 New Development

Small development groups tend to have efficient and simple organizations. Their software development process is likewise simple during the initial phases of the project. It is typically divided into three phases:

Phase	Activities
Programming	The application and its components are created. Deliverables include analysis and design documents, code, and user documentation. A preliminary test of individual components, called unit test, is performed during this phase.
Integration test	The correct interaction of all system components is tested against requirements. Code modifications necessary to correct deficiencies continue until successful test completion.
Release	The tested system is released to the customer. The delivered system is identified with a unique version identifier. From now on, it is vital to be able to determine all components simply by referring to this version identifier. Further modifications to the code are suspended.

3.4.2 Maintenance

The above scheme is applicable when the software for a certain project is written for the first time. As soon as the code is released for use in production, maintenance activities, such as solving any problems pointed out by the users, begin.

Additionally, development begins for a new version of the application. New versions are usually required to implement enhancements to the product functionality or to port the whole application to additional hardware platforms.

At this time the initial project of low-medium complexity has become a number of common projects to be simultaneously supported from the same development group. This change will be reflected in changes to the software development process and configuration management requirements. The code must be managed with respect to two main issues:

- The need to share code for reuse between successive versions of the application
- The need to control and manage the integration of modifications to the code in these several versions.

3.4.3 CMVC Concept of Process

CMVC allows the customer to decide which CMVC functionality is required during the current project phase and simply add more CMVC functionality as demands change. This is accomplished by a mechanism CMVC refers to as configurable process. The CMVC process is defined when creating CMVC components and CMVC releases. CMVC component or release processes can be changed at a later date.

CMVC provides a basic process scenario for defect and feature tracking, as well as for version control and release management. Customers can tailor their implementation of this by defining their own process as a combination of subprocess options offered by CMVC. They choose their process definition from a list of choices provided by CMVC or create new choices of their own. Typically, small-scale users of CMVC will find the predefined choices suitable, as they have fairly generic labels and cover most combinations.

By manipulating process selections for their CMVC components, small-scale CMVC users can determine whether or not, and how, defects and features will be processed. By manipulating process selections for CMVC releases, customers can determine whether defects will be tied to version control and to what degree release and build management will be managed.

3.4.4 Defect and Feature Tracking Options

Version control can be done without problem tracking in CMVC, but if problem tracking is desired, version control will be tightly linked to defects or features identified by CMVC users. Likewise, defect tracking can proceed without release management support, or it can be integrated by means of CMVC tracks. Most small-scale CMVC customers believe that version control over development data files is an immediate requirement as soon as the project begins. There is great variation in their sense of when to begin and how to perform problem tracking, however.

Some small-scale projects have defects from the very first day, whereas others introduce them much later. If postponed, defect tracking might begin at one of these milestone events:

- Beginning of the first formal testing (unit test)
- Beginning of integration test
- After the production version has been installed.

There is also variation in when customers choose to begin using feature tracking, and how they process features. Some save feature tracking for the maintenance phase, eschewing features during earlier phases. Some will never use the DSR subprocess for defects but always use it for features.

Finally, these customers make use of different process selections to govern components that manage different types of development data. Design material, end-user documentation, and code, for instance, may not all require the same processing with CMVC.

Continental uses problem tracking only after an application goes into production and to manage the software changes arising from maintenance activities. Thus newly created components related to a particular application will not have any defect subprocesses selected. At some later time, these component process selections will be changed.

BT Laboratories uses problem tracking from the earliest stage of the development process but only uses CMVC defect processing for significant defects. The company mixes CMVC problem tracking with a preexisting manual defect process to track all proposed defects. A person, designated as the Change Manager, reviews the proposed defects and decides to enter them as CMVC defects if their resolution is likely to take more than 10 days to complete. Insignificant defects are presumably given a fix or no-fix judgment and not tracked further. If entered into CMVC, the defect will be managed under both the DSR and verify subprocesses. This choice is designed to keep a close eye on how unplanned development activities (for when is a defect planned?) impact the delivery schedule.

SCS uses a predefined component process selection, *prototype*, for components managing documentation files. The hardware testers use CMVC purely for the problem reporting system because the hardware development, done at another location, is not under CMVC control. SCS uses the more stringent predefined process selection, *development*, for components managing source code. Its intention is to move to the predefined process, *maintenance*, after the formal testing completes and its application moves into its production environment.

3.4.5 Release Processing Options

If defect and/or feature processing are in place, release processing can be implemented in several ways. Release processing is defined by the customer as a specific combination of release subprocesses. As with defect processing, CMVC provides a set of choices of predefined combinations of these subprocesses. Most small-scale users of CMVC find that these predefined release process combinations meet their needs; however, CMVC users can define their own combinations.

As with defect processing, release processing requirements usually evolve over time on a small project. Initially, these customers will not use the track subprocess, but as the development effort reaches the integration phase, most CMVC

customers will begin to use it. Once the track subprocess is selected, most small-scale projects will use it in a fairly simple manner, utilizing fewer than the full complement of release subprocesses offered.

Some groups at BT Laboratories, for instance, does not use the track subprocess until the development baseline reaches a significant size and stable state. At that time, it begins to use only level and fix subprocesses.

SCS has defined its own process combination, which consists of the track, release, fix, and level subprocesses. SCS anticipates that as its development effort proceeds and its familiarity with CMVC deepens, it will find the configurable process to be a valuable feature in CMVC which they can better exploit.

3.5 Management Reports

By providing project management information through the customized report facility, CMVC becomes a valuable tool for the project manager who needs to monitor the pulse of the project. The report function of CMVC allows users to gain access to file and configuration change history, software metrics related to defects, project status data, and many more types of information. This information can be formatted or further processed in many forms in conjunction with simple programming or familiar UNIX utilities such as **shell**, **awk**, **sed**, and **PERL**.

CMVC inherits its powerful reporting capabilities from the underlying relational database. CMVC provides not only SQL style query capability but also predefines tables and views to use with CMVC report commands. *Ad hoc* reports are accessible from the GUI, where the user is prompted with fields appropriate for each record and allowed to select ordering of the results by these fields. The command line report utility, however, provides additional flexibility in combining CMVC reports with other UNIX commands for additional manipulation of the data.

Most small-scale projects do not use the reporting facility to near its capacity, but all appreciate its potential. One reason for this is simply the size of the projects and the limited ways in which the customers use CMVC at first do not make for complicated reports. For instance, on a 20 person project, how many different managers need frame queries specifically to extract information on open defects in components owned by only their developers for which there are tracks in a particular release? Likewise, if there are only 15 open defects and only one release, there is not likely to be a need for a report on only those defects opened by a particular user in a particular release, ordered by the phase found field. A simple scanning of the report on all defects will identify the desired data when the data is small in volume.

All of the small-scale CMVC customers report frequent use of the GUI for *ad hoc* report generation, however. And, they report that this usage goes up dramatically, as their projects reach critical development events such as completion of integration testing. Finally, these CMVC customers devote as much attention to their decisions on modifications to make to choices lists and potential configurable fields as customers who use CMVC on a larger scale, because they anticipate the later need for historical data accumulated in CMVC's repository.

3.6 CMVC Support for Heterogeneous Hardware and Operating Systems

CMVC is available on the best selling UNIX platforms (HP, Sun, and IBM RISC System/6000) and for IBM's OS/2 and Microsoft's Windows on PCs. Small-scale projects appreciate CMVC's support of these heterogeneous platforms and operating systems and its taking advantage of common open system standards such as TCP/IP and NFS.

Flexibility is highly valued in a small-scale project where few people often perform multiple roles on the project, and where platform choices may be driven by a limited budget or factors beyond their control. There are several reasons why flexibility in hardware and operating system is significant to this type of CMVC customer. Reasons cited during this study include the fact that CMVC provides them with the ability to leverage their:

- Preexisting platform-specific skills
- Common CM skills across platforms
- Standard CM policies and CM procedures across projects
- Preexisting or planned new development and target platforms
- Centralized configuration management control with remote access
- Ensured growth path
- Integrated defect processing and change control

3.6.1 Preexisting Platform-Specific Skills

Availability of CMVC on divergent platforms ensures that small-scale CMVC projects can leave users on the platforms they are most familiar with, or on which their other tools are currently hosted, and still give them CMVC access through local CMVC clients, or remote client execution and NFS-mounted file systems.

3.6.2 Common CM Skills across Platforms

Although there are some fundamental differences in system administration or GUI standards and conventions between one hardware platform to the next or between one operating system to the next, CMVC skills transfer readily among them. This provides these CMVC customers with the flexibility to transition users from lower performance development platforms to better ones as the price-performance ratios change and new equipment is purchased with minimal penalty in transferring their CMVC skills.

3.6.3 Standard CM Policies and CM Procedures across Projects

One of the most common requirements for a configuration management system is availability on different hardware platforms to ensure that current and future projects can take advantage of a single CM tool. At Continental a tool to be adopted as a company standard should fit in the current as well as in the future hardware configuration.

3.6.4 Preexisting or Planned New Development and Target Platforms

The requirement for CMVC to fit in with preexisting development or target environments is particularly important to a customer like BT Laboratories. Its development and target platforms include the same UNIX platforms supported by CMVC today, and the company can install a single CMVC server that can be accessed from the different UNIX clients across its LAN.

In some cases, BT Laboratories ports applications from old UNIX platforms to new platforms, needing to maintain releases appropriate for both. It also needs to know that should it need to port to additional UNIX platforms, CMVC will support them.

3.6.5 Centralized CM Control with Remote Access

By implementing a client/server model CMVC is well suited for centralized control of multiple, often remotely located small development projects. At BT Laboratories and SCS the development and test teams consist of about 20 people operating on one LAN. Because the applications are built to run at different sites, defects may need to be registered remotely.

3.6.6 Ensured Growth Path

The customers visited believe that CMVC is capable of growing as their projects grow. CMVC is able to support both small, tight workgroups now, and large multisite organizations working on common projects later. At BT Laboratories and SCS the development and test teams believe that the maintenance activities coupled with further development activities will expand their CMVC configuration.

3.6.7 Integrated Defect Processing and Change Control

By providing functions like configuration management, change management, and problem tracking integrated in one system, CMVC offers the capability to perform all of these tasks in a single development environment. SCS and the other customers discovered that this functionality was unique on the market when they conducted their marketing trade studies.

3.7 Rolling Out CMVC on New Projects

Customers who use CMVC for small-scale projects introduce CMVC gradually into their organization. Typically, they bring one project at a time to CMVC and, after having worked out any difficulties with that project, go on to the next. Depending on the business characteristics of the individual companies and the phase in their development efforts, they take advantage of the CMVC functionality in different ways.

3.7.1 Setting Standards, Conventions, and Policies

Introducing a new tool of any kind in an existing development environment always involves a little bit of planning. No customer wants to face turbulence of any kind that could interfere with team productivity or risk any kind of acceptance problems. With respect to CMVC, it simply means getting the functionality you need with a tool that fits into your development process without creating unnecessary overhead.

With this goal in mind, Continental put some resources into planning the usage of CMVC tailored to its needs. This included mapping the company standards, such

as naming conventions and the software development process, into CMVC component structure and user roles.

3.7.2 Limited CMVC Functionality at First

Continental decided to use only the configuration management and change control functionality for the first development cycle and then introduce problem and feature tracking for the combined maintenance and successive development phases.

SCS adopted a different approach. It made the choice to use a system with integrated configuration management and problem tracking functions and started immediately using full CMVC functionality (although it does not use all possible subprocesses). Specifically, SCS chose to bypass DSR subprocessing of defects and the approver subprocess to keep it as simple as possible. The microcode developers use CMVC to track hardware problems at first, but they plan to use CMVC for version control at a later stage.

3.7.3 Initial Education and Consulting

Other aspects closely related to introducing a new tool are product installation and customization and user education. In all of the small organizations we visited the person responsible for CMVC administration is the person in charge of system, database, and family administration. The initial setup of CMVC was done working with an IBM person with a sound background in CMVC concepts. This preparatory work has been reflected in the overall CMVC acceptance because the developer smoothly interacts with a well-defined environment.

This approach has been followed for the subsequent user education. A subset of the original CMVC documentation has been used to prepare presentation material targeted to the different CMVC roles. At BT Laboratories and SCS the user training is mostly done on the job, whereas the written and online documentation is used for reference.

At Continental a well-detailed set of user manuals provided by the software engineering group substitutes for the original documentation for the day to day work with CMVC. Each manual is targeted to cover all of the tasks related to a specific CMVC user role. The valid user roles at Continental are described in the next section. This collection of manuals:

- Offers an overview of the different activities related to predefined user tasks and their mutual interactions
- Defines the development environment for each CMVC role
- Stresses the importance of following predefined naming conventions for login and path names
- Explains in detail each one of the single steps the developer may go through during his or her work with CMVC in a straightforward question-and-answer format
- Includes the screen captures related to each single step, starting with the customized CMVC Task window.

3.8 CMVC, Role Specialization, and Project Organization

CMVC allows definition of specialized end-user roles. Each role is defined according to the CMVC actions it can perform. Access control within CMVC is based on user roles and extends well beyond the concept of read-write-execution permission provided by other CM systems.

3.8.1 Development Role Specialization at Continental

At Continental the software engineering department is the organization responsible for defining and refining the companywide standards for the software development projects. The goal of this effort is to improve the quality, the development process, and the maintainability of the software. To achieve this goal, it is extremely important that the development team operates in well-defined roles where the different tools are configured to fit the needs of the roles.

Continental has defined three roles within a project (see Figure 2):

Software developer The developer works within a determined development and/or maintenance environment in the phase programming. Within this environment he or she works in the context of a module, that is, the developer creates and modifies individual components of a module where a module represents a group of programs to implement a certain function. A private work area, called the module area, is provided for the developer. He or she owns all privileges in his or her module area and has read access in the release area.

Software manager The software manager works within a determined development and/or maintenance environment in the phase integration test. The software manager is responsible for the preparation and accomplishment of the integration test. He or she rejects defective components to the responsible software developer back into the phase programming. The software manager performs CMVC administrative tasks such as creation of modules (that is, CMVC components), assignment of modules to software developers, creation of CMVC releases and levels, CMVC user administration, and administration of the directory structures for the integration test and for the released version of the product. He or she owns all privileges in the integration test area and has read access for the release area.

Build engineer The build engineer works within a determined development and/or maintenance environment in the phase release. The build engineer is responsible for the release build. His or her work area is called the release area. Before delivery he or she assigns a unique version number to the system according to the company naming standard. He or she owns all privileges in the release area.

3.8.2 CMVC User IDs and Project Roles

According to this organization an AIX login is assigned to a CMVC user to make sure that all of the system variables are set accordingly. CMVC users use different AIX logins when performing their different roles and dealing with their different points of view on the project data through CMVC.

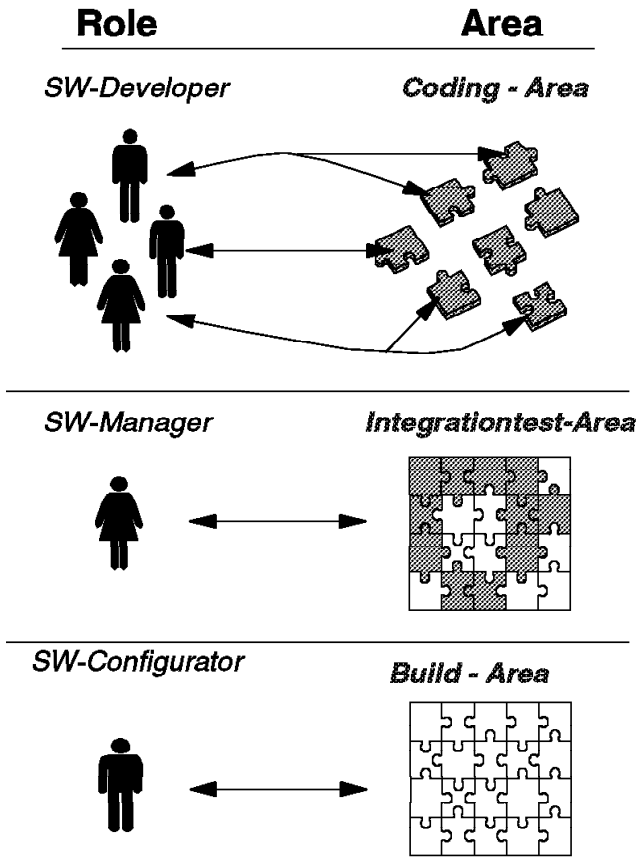


Figure 2. CMVC Roles at Continental

3.8.3 Development Role Specialization at SCS

A completely different approach has been put in place at SCS. Each team member has been defined as a CMVC user with Developer+ and Component Lead authority, an approach SCS believes is best suited for a small group of experienced people.

The approach BT Laboratories has chosen to address the CMVC roles is somewhat inbetween, making use of more differentiated profiles.

3.8.4 CMVC System and Family Administration

One common feature at the different customer sites is the use of one person serving as system and family administrator. This person provides the following support to the development team:

- Installs CMVC
- Gives assistance to a new project that is rolling out CMVC
- Defines and accomplishes a backup strategy

- Enhances the system with ongoing customization, such as user exits, predefined reports, and customized task windows
- Supervises the results of the build process, as at BT Laboratories and SCS.

3.8.5 Access Authority Groups

CMVC lets you choose these roles in the default configuration files, known as access authority groups. The names and their associated subset of CMVC actions are meant to be used as guidelines to configure your own system, and each customer has taken the opportunity to do that as shown above.

3.9 Applicability

Customers use CMVC mainly to support software development efforts. Nevertheless they recognize that CMVC is also appropriate for controlling a wide variety of data, including but not limited to source code.

3.9.1 Variety of Programming Languages

At all customer sites CMVC is tightly integrated in the development environment for storing and controlling source code and operating system command language scripts. CMVC is appropriate for controlling any source code language, from assembler to 4GL to interpretative languages. The C language is common to every UNIX development. Depending on the type of application, other programming languages are still better suited for writing special functions.

For example, Continental uses APT, a 4GL language for the Sybase RDBMS, and at BT Laboratories some code is written in C++. At SCS the microcode for the watches is written in a 4-bit assembler.

3.9.2 Application Documentation

Documentation and design documentation are not yet commonly stored in CMVC, with the exception of BT Laboratories, where the design documents are stored together with the code for a certain code release. These documents are not subject to any changes, at this stage. Logically, they belong to a certain version of the product, and in this way they can be retrieved together with the code. This approach is then useful for auditing and quality control purposes, because one system keeps track of the link between the design specifications and the software that is supposed to meet them.

In the case of Continental the documentation is written on PCs using word processing programs under DOS Windows. The availability of the CMVC client for this platform will be a strong incentive to put the documentation as well as the code under CMVC control.

3.10 Component Hierarchy

The gradual introduction of CMVC is reflected in the component structure of the CMVC component hierarchy (see Figure 3). It initially maps the existing directory structure for the code, so that no restructuring of the code organization is to be seen as mandatory in the first place. Nevertheless, CMVC can manage complex code organization such as multiple releases and levels, so the growth of the project

will lead to transforming flat structures into more complex interrelated graphs. The following examples show the initial project setup at Continental.

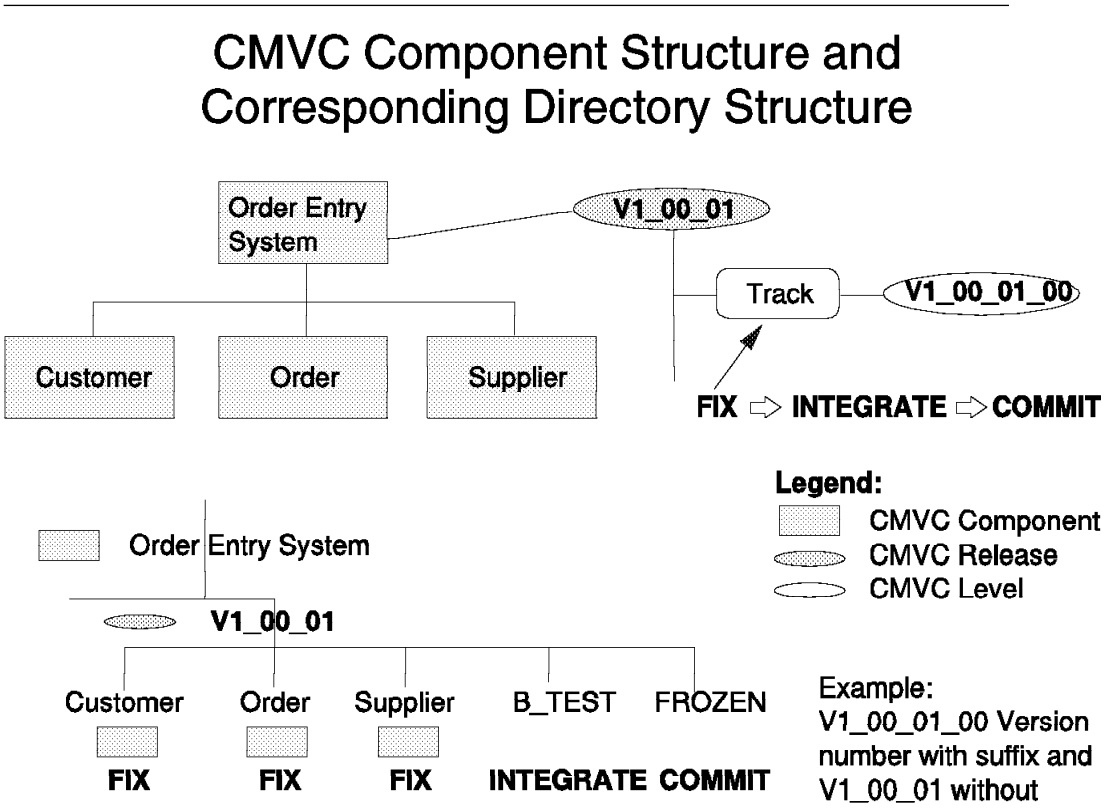


Figure 3. Relationship between CMVC Component Hierarchy and Directory Structure

3.11 ISO 9000 Certification and CMVC

Throughout all industries, software development organizations are focused on improving quality. This concern is common and highly sensitive to all of the customers participating in this study, even those whose application development projects were fairly small-scale. The ISO 9001 certification also has become a related critical business issue for those industries that want to provide services based on software either directly to the market or as subcontractors for bigger projects. This is particularly true for Europe, but the necessity of ISO 9000 certification is increasingly becoming an issue for companies operating in North America and the Asia Pacific region also. BT Laboratories and SCS expressed this sentiment.

Many customers see CMVC as the tool for implementing a quality management system for the tasks involved in software development and helping them meet their goals in complying with some key elements of ISO 900.

BT Laboratories well-defined manual process raised some concerns during a first audit for ISO certification. In response, the company introduced CMVC in such a way that it mapped directly to its manual process. In doing so, it gained an overall traceability of its processes and subsequently passed the audit for ISO certification. The company believed that CMVC enabled it to introduce automation that

supported its existing development process and therefore contributed significantly to achieving its important goal of ISO 9001 certification.

Chapter 4. Use of CMVC on Medium-Scale or Companywide Basis

This chapter shows the usage of CMVC from the perspective of medium-scale locations and company installations. These companies use CMVC in a sophisticated way, put different kinds of data under CMVC control, rather than just source code, use CMVC on different development and for different target platforms, and use CMVC as an integrated tool in a network environment. They have developed several tools around CMVC which they use to extract data from other tools and products into CMVC and get data out of CMVC for further processing by other tools such as report or graph generators. The chapter explains the impact of this sophisticated environment on the overall administration and describes the solutions for introducing CMVC into a company's development processes. This chapter also demonstrates how CMVC is used together with other software engineering paradigms like object-oriented analysis (OOA) in contrast to the traditional waterfall model.

4.1 Representative Customers

We visited two sites that are representative of a medium-scale CMVC location:

- The development site of the Intelligent Services Platform division of MCI in Colorado Springs, Colorado, USA.
- The development location of the Storage Systems Division of IBM Tucson, IBM Tucson, Arizona, USA

These two locations have common characteristics with respect to their size and CMVC usage.

4.1.1 Similarities of CMVC Usage

Both locations have used CMVC since 1992 and since then have put multiple families and projects under CMVC control. As of the time of writing this book, about 10 families are active in both locations, and the files under CMVC control number in the tens of thousands. Appendix A, “Customer Profiles” on page 101 summarizes the most important characteristics for these CMVC development locations.

The two sites have come across similar challenges with respect to using CMVC in a heterogeneous network of clients and servers, and they both use CMVC to develop software for different operating system platforms. Both locations had similar experiences as their user community grew over time. In response, both have set up separate departments for administration and end-user support. As the number of projects increases and the number of users crosses a threshold of several hundred, CMVC becomes mission critical to the entire success of the development location. Both sites therefore have incorporated CMVC as an integral part of their development processes and approaches.

The characteristics of these locations are rather different from the installations as described in Chapter 3, “Use of CMVC on a Small Scale or in the Initial Phases of Development” on page 27. Once a site reaches the size of these installations, additional requirements from the users and additional administration effort come up almost immediately. To support that number of users and the size of these

installations, there have to be dedicated support people that maintain, customize, and administer the CMVC installation. In addition, this support staff is dealing with issues that arise when you integrate CMVC into a development process that is already in place.

Later sections of this chapter describe how this integration is done and the impact of the integration on the overall development approach.

However, different projects tend to choose different paths within a defined development process. Some even decide to chose different tools or processes and want to integrate a configuration management tool such as CMVC with these modified approaches. Both sites show that different projects within one location may use different processes and integrated tools, but that CMVC is flexible enough to be used in these different environments. Thus there is no standard development process in these locations, but the different groups and projects define their own customized development approach to meet their different requirements, and CMVC is used in different ways to meet the requirements in the different projects.

It is important to understand that CMVC is not based on a predefined development process, but that it can be used within certain limits with various development paradigms.

4.1.2 MCI (Colorado Springs)

MCI is the second largest telecommunications company in the world and offers various kinds of telecommunication services to its customers. The networking center installation of MCI is located in Dallas, Texas, and multiple data centers throughout the USA offer the support for the different geographic regions. The site of MCI represents a customer that uses CMVC to develop software which is an integral part of the company business.

The development site that was visited is in Colorado Springs, Colorado, where the Intelligent Services Platform Organization develops software for telecommunication services. A second development site of MCI is located in Cedar Rapids, Iowa, and the software development focuses on application areas such as video reservation systems, fraud detection, or other similar telecommunication systems. The group that was visited is organized into about 50 different projects with a total of about 350 developers.

This group had worked with other home-grown configuration management products before but ran into problems as there was no integrated problem tracking with the associated code control. The maintenance of different home-grown configuration management tools became unacceptable, and the introduction of new hardware and software platforms did not go along with proper support from these tools. It was therefore decided to replace this variety of configuration tools with CMVC within a transition period of a few years.

Since the introduction of CMVC in 1992 with the first sample project, CMVC has become the widely accepted standard configuration management tool within MCI, and almost half of the development projects are already using CMVC. The first installation was done with CMVC Version 1.1, in the meantime MCI migrated first to Version 2.1 and is now operating at Version 2.2. By the end of next year CMVC will be used by all the projects, and the Colorado Springs development site will be a pure CMVC shop.

In parallel, based on the positive experiences from Colorado Springs, the MCI site at Cedar Rapids is also introducing CMVC. Whereas Colorado Springs is using ORACLE as the underlying database, the site in Cedar Rapids is using IBM's DATABASE2/6000 (DB2/6000) to store CMVC's control information. MCI uses IBM RISC System/6000 Models 990 and 590 as CMVC servers, and many of the developers access CMVC through clients operating on various platforms such as HP, SUN, DOS/Windows, or OS/2. As of the time of writing five families are administered in the Colorado Springs site with about 160 active user accounts.

The development tool environment consists of CMVC along with SDE Workbench, along with INTERLEAF for documentation purposes and compilers for C and C++. In addition, some projects use AIX Interface Composer (AIC) Version 1.2 and CADRE TeamWork for analysis and design. Other sections in this chapter describe the unique features of the MCI site, as they are typical for a medium-size CMVC installation.

4.1.3 IBM Tucson

IBM IBM Tucson represents both a hardware vendor as well as a software development site. In fact these functions are more or less independent of one another, so it is not surprising to see different approaches as far as the use of CMVC is concerned.

4.1.3.1 Programming Center

The software development organization is called the *Programming Center*, and its main purpose is to develop program products for the worldwide market. The most important product developed here is ADSM (ADSTAR Distributed Storage Management), which is a backup and archival software running on multiple server platforms such as MVS, VM, AIX and OS/2. In addition, several ADSM client platforms are supported, such as PC/DOS, Microsoft Windows, SCO UNIX, HP/UX, NetWare, DEC ULTRIX, OS/2, AIX, and Macintosh platforms.

The Programming Center uses CMVC to control all code for all of the platforms mentioned above. A two-person central support is in charge of the CMVC administration and support issues. This is necessary, as ADSM is developed not only in Tucson. Rather, there are development locations in Mainz (Germany), Haifa (Israel), and San Jose (California, USA), which are connected to the Tucson LAN through an IBM internal wide area network using different network connections. For example, Mainz is connected right now through an intermediate SNA line to Great Britain, and from there through a direct TCP/IP line to the USA.

The CMVC server in Tucson is a RISC System/6000 Model 560 running CMVC Version 2.2. It uses INFORMIX as the underlying database system and PVCS from Intersolv as the version control system. PVCS was chosen, because it was already in use for library management stand-alone, so the licences and the skills were ready for immediate use.

The developers and testers access CMVC through clients running on AIX and OS/2; in addition they use an IBM internal CMVC client that executes on VM. The installation is based on a single CMVC family, which, due to the size of the ADSM project, encompasses quite a few files. Right now CMVC controls more than 15,000 active files, which are worked on by more than 300 active users. These CMVC user communities are spread worldwide, and the number of transactions reaches a limit of more than 4,000 per day.

4.1.3.2 Tape Drive and DASD Development

In addition to the Programming Center, other organizations in Tucson use CMVC, and these are centered around the development of tape and disk drives and their control units. These organizations need to develop hardware as well as software and microcode, and, as with the Programming Center, their CMVC installation is quite large and geographically distributed.

The development effort for DASD controllers is done in cooperation with a development site in San Jose, California, and the CMVC server is run there. This project uses CMVC Version 1.1.2, and the Tucson site accesses the CMVC server in San Jose through its local CMVC clients, which mostly execute on AIX and OS/2 and are connected with San Jose through a TCP/IP wide area network. The CMVC server in San Jose is also used to share code with other control unit development projects in Tucson. IBM Tucson also runs several local development projects, which use a local server running CMVC Version 2.2 as the configuration management tool.

IBM Tucson uses SCCS as the version control tool underneath CMVC, and the development processes and approaches differ among the various projects. Some have come up with a new development paradigm and use OOA methods along with the CADRE TeamWork product.

To support the groups for the tape drive and DASD development, there are seven families in the CMVC server, which is a RISC System/6000 Model 550 using Oracle as the database management system. This server controls more than 10,000 files, and it is accessed by more than 100 users. This results in a daily workload for this server of about 5000 CMVC transactions.

4.2 CMVC in a Network of Clients and Servers

When the development environment becomes heterogeneous and multiple target platforms have to be supported, the maintenance of common source code becomes even more important. CMVC supports this network scenario by offering multiple clients in various operating environments which still access the central CMVC repository on the AIX server. Both MCI and IBM Tucson develop on a heterogeneous set of operating systems, and they both use a variety of CMVC clients during their development activities. In addition, the developed code is targeted for multiple target platforms, whereas the source is still maintained on one environment, namely AIX. The sections that follow show the usage of CMVC in this network environment.

4.2.1 Clients on Several Platforms

The hardware used by the CMVC server and the hardware used by the various end-user groups can differ from one another. The platform from where CMVC is used may be a question of convenience, and a user may use CMVC from the operating system with which he or she is most familiar. In fact, the platform of choice is selected on the basis of different criteria, and examples of these selection criteria are explained in the following list:

- A *code developer* wants to access CMVC from his or her development environment, and this can be either the target platform for the final software or (in case of open systems) a compatible platform.

For example, the ADSM development group at IBM Tucson uses clients running on local AIX systems, which access the CMVC AIX server. Developers at MCI use the DOS/Windows client of CMVC when they work on projects operating in the DOS/Windows environment. MCI developers use the CMVC clients running on SUN and HP when working on projects that operate in these environments.

Sometimes a client is not yet available on the development platform, such as VAX/VMS or other UNIX platforms. MCI mostly uses high-end PCs as the workstation hardware of choice and runs TCP/IP and X11 emulation under OS/2. Thus it can use the PC as an X-Station to run any other X-clients from a connected AIX or UNIX machine. However, rather than running CMVC as an X-client across the TCP/IP network it is much better to use the local OS/2 client of CMVC. It is not surprising then, that MCI views the arrival of the DOS and OS/2 clients as extremely important for the company use of CMVC.

- A *documentation developer* wants to access the client on the same platform that is used for developing the documentation.

  For example, the *information development* organization at IBM Tucson, which develops the publication material for ADSM, uses the IBM BookMaster product running on the VM operating system. It uses CMVC as the problem tracking system and has a specific component within CMVC that it uses to report CMVC defects. It is only natural that this group uses a CMVC client running on VM to access CMVC (the VM client is available as an IBM Internal Use Only tool). Likewise, MCI uses Interleaf for documentation purposes in some projects, and therefore uses the clients on AIX, which is the Interleaf platform.

- A *tester* needs to access CMVC from the environment he or she is using for running the tests. For example, if particular test tools are used in a specific environment (for instance, running recorded test cases during a regression test phase), CMVC needs to be available from the corresponding test environment. This is particularly useful in the OS/2 and DOS/Windows environments. Also, testers might want to use CMVC in the software environment with which they are already familiar, rather than having to learn a new operating system with a new user interface.

  For example, the ADSM test groups are used to work with host-based operating systems such as VM. They use the CMVC client on VM for problem tracking and reporting, even to report defects that they encounter during the test phase of the ADSM products on AIX.

- A *manager* uses CMVC from the platform he or she is using during most of the day. In the case of IBM, most internal utilities and communication are based on VM/CMS, so managers are most familiar with that operating system.

- An *administrator* uses CMVC mostly from AIX, and sometimes even from the command line rather than the GUI.

4.2.2 Code Developed for Multiple Platforms on Multiple Platforms

One of the nice features of CMVC is the support for multiple target platforms from one CMVC server platform. Many development shops face the problems that the software they write runs on multiple operating system platforms. This can imply that one product runs on multiple platforms or that several projects each operate in a different environment, and they all use CMVC as the central library product. This includes not only those platforms that are derivatives of UNIX (like AIX, HP-UX,

SunOS) but also proprietary operating systems such as DOS, OS/2, VM, MVS, and Macintosh.

A good example is the MCI development site in Colorado Springs, which develops software mostly for its AIX and VAX/VMS systems, but also for OS/2, Stratus/VOS, SunOS, HP-UX, PC/DOS, Windows/NT, and others.

The ADSM development group at IBM Tucson has to support all of the client and server environments that are supported by ADSM. Currently these are PC/DOS, AIX, OS/2, HP-UX, Novell NetWare, SCO 386 UNIX, SCO Open Desktop, SunOS, Solaris, DEC ULTRIX, Macintosh, and DOS/Windows for clients, and MVS, VM, AIX and OS/2 for servers. The list shows that not only are multiple operating systems supported but also different systems from software vendors such as IBM, Hewlett-Packard, Bull, SUN, Apple, Novell, and SCO. All of the ADSM code for all of these environments is controlled by a CMVC server running under AIX.

Within CMVC, the components are structured according to a kernel/shell paradigm, which implies that platform dependent code is put into different CMVC components than the code which is common across platforms. When extracting a release for a specific target platform, all files belonging to code maintained in common CMVC components go along with the code that is stored in CMVC components specific for that platform. Also all files required for the build on that platform are extracted onto the target platform.

The extraction across the network is done using a TCP/IP connection. IBM Tucson either uses an extract across NFS-mounted file systems or extracts to a local file system on the CMVC server and then uses FTP to transfer the files over to the target system where the build procedures are run. The files required for the build procedures depend on the corresponding target operating system. For example, to support VM, a set of REXX EXECs are stored under CMVC. The build procedures are then executed on the corresponding target system to build the system.

MCI discovered a performance issue due to a problem in the underlying TCP/IP software when it tried to extract to a remote file system that was NFS-mounted on another machine. The time needed to do this remote extract was pretty slow, so it decided to do a local extract instead and then use a copy command to copy the files to the target system. By doing so MCI can drastically reduce the cycle time required for the build.

MCI also had to solve an interesting problem for a non-IBM platform (Stratus), which allows a different syntax for file names than the syntax used on AIX. This requires that the file names between the Stratus and the IBM platform be mapped before they are exchanged and that special characters in file name, such as the dollar character, which are valid for the Stratus/VOS operating system, but invalid for AIX, be eliminated. MCI implemented an elegant way of doing this by providing a user exit, which is called for check-in, check-out, and release extract operations. This user exit maps characters that are invalid for AIX (such as the dollar character) to other valid special characters (such as an underscore character). Also, a configurable field is used to store information inside CMVC about the target directory to which a file is going to be extracted. A configurable field is set up in the user table of CMVC for all users working on the Stratus project. Whenever a file is extracted to the Stratus platform, this field is checked, and, if it is not empty, the file is extracted to that target directory.

4.2.3 Networking and Remote Access

When the development group reaches a certain size, as is the case with MCI and IBM Tucson, the development effort is no longer a question of a local area network. Instead, the development groups are connected through wide area networks with other development locations, which also need to have online and fast access to the development environment in the corresponding remote locations. Those who need to use CMVC interactively have to have an immediate response from the system, and the access across the network must be nearly as fast as the access within a local LAN or between LANs connected through a high speed bridge.

When introducing a central repository for source code maintenance into a company, concerns may arise about the ability to have high performance access to a central data base across networks. This is not only an issue for very large scale CMVC installations such as described in Chapter 5, “Use of CMVC on Very Large-Scale Basis” on page 75. Even CMVC installations in medium-scale environments, such as the MCI development location and the IBM Tucson site, are being confronted with these questions.

For example, MCI in Colorado Springs is mainly a local area network with its big CMVC server. However, the various customer service centers distributed all over the United States need to have immediate access to the defect tracking functions of the CMVC system. Also, the second MCI development location in Cedar Rapids, Iowa, is connected to MCI in Colorado Springs through a wide area network link. Although the two sites are operating mostly independently of one another, sometimes the need arises to have a cross system access for administration purposes. MCI has reported no performance issues with its network environment.

An even more widely distributed use of CMVC in a network environment is found at the IBM Tucson location. As explained in 4.1.3, “IBM Tucson” on page 47 two different CMVC user groups can be found in Tucson:

1. The Programming Center develops ADSM and operates with a CMVC server system in Tucson. Various other development groups located in San Jose (California, USA), Endicott (New York, USA), Mainz (Germany), and Haifa (Israel) operate with this server in Tucson.
2. The DASD and Tape Drive Development group in Tucson works with a similar group located in San Jose and accesses the CMVC server located in San Jose.

The CMVC end users are connected in a local LAN, which has proper TCP/IP links through an IBM internal network to the other U.S. locations. The TCP/IP connection to Germany is more complicated, as the network connection is just enabled to Great Britain, and the connection between Germany and Great Britain is through a 64K SNA connection.

Whereas the connection within the United States is almost instantaneous, the remote connection to Germany is too slow to be of practical use especially when extracting files and a lot of data has to be moved. Instead, Mainz does a local extract to a local directory on the server and then uses `FTP` to transfer the file to Germany. When direct access is required, the remote locations take advantage of the command line interface, which also allows shell scripts to be built when the same operation has to be applied against a couple of files at once.

4.3 Evolution to a Company Standard

Software development organizations that have been in this business for some time do not start with CMVC from day one. On the contrary, many companies and organizations have started without a specific tool in place for configuration management and version control, because it was not considered important and required at the very beginning. However, at some point in time the management problems became apparent, and tools and procedures had to be put in place to solve them. This section describes the evolution of the introduction of CMVC into the development process at MCI and IBM Tucson and shows the rationale and benefits behind that evolution. It is a typical scenario that many other companies have gone through along the path to configuration management.

4.3.1 Home-grown Systems and Their Problems

Before MCI used CMVC in Colorado Springs, it used various home-grown configuration management systems. However, as the development activities became larger, several problems became apparent that MCI could not solve with the systems in place.

MCI became involved in working with multiple hardware and software platforms. However, there was no way of providing an integrated development approach across multiple platforms, because the development groups on each platform could not access code or report problems directly for projects working on other platforms. The lack of cross-platform support in the CM tool was the cause for the following problems:

- As each project used its own way of problem tracking, there was no standard across projects. Each time developers joined a new project they had to learn new procedures, tools, and processes in order to work effectively. This additional learning effort was expensive and ineffective.
- The systems in place did not allow change control. They were based on native UNIX tools such as SCCS, but there was no integrated approach to connect problem tracking and code changes. It was impossible to reflect code modifications back to particular problems.
- As the different projects used different home-made tools and procedures, additional effort had to be spent to develop and maintain those private tools. The maintenance of these tools became a problem, as the developers of these tools were also part of other development teams.

A developer summarizes his experiences with the home-grown CM tools that were in place as “...just awful....”

The Programming Center at IBM Tucson used to work with host based CM tools before CMVC was introduced. When they faced the requirement of having to develop software for the workstation operating systems, they soon realized that it was not a good idea to keep the central code repository on the host system. The multiple set of client and server platforms supported by CMVC was the key factor for them to select CMVC as the CM tool for all platforms including the host operating systems like VM and MVS

4.3.2 Introducing CMVC into the Process

In 1992 MCI decided to try CMVC in a sample project. From its experience in the past it saw a need for a standard CM tool that provided integrated problem tracking and version control across multiple platforms. It chose to use CMVC for one of its most critical projects at that time. The sample project was already a year behind schedule. It was well suited for a configuration management product such as CMVC because the project was being worked on in different development locations. The number of 500 open defects at the time CMVC was introduced illustrates the critical situation of that project.

MCI bought CMVC along with Oracle as the underlying database. It first put that sample project under CMVC control, and shortly afterwards the Intelligent Services Platform Organization made the strategic decision to use CMVC for all projects. It extracted the current level of the files from its existing CM libraries and loaded them into CMVC as a new baseline.

New projects started to use CMVC, and by chance CMVC was used on the non-AIX platforms (like HP-UX, SunOS, Solaris, VAX/VMS) first. At the time of writing that organization has at least one project on every platform under CMVC control, and by the end of 1995 it "...will be a total CMVC shop."

To solve the additional support required by increasing the numbers of CMVC users, a separate department was put in place to provide development services such as LAN support, system test coordination, project management, budget planning, and configuration management. At the time of writing this group is working on a system methodology description for development activities in the low-end and mid-range system area. CMVC will be part of that methodology, and it will be the configuration management and version control product of that entire division within the next year.

To help new projects getting started with CMVC, the CM group has developed specific education material for new users. The CM department also helps new projects to define the CMVC administrative setup, supervises, and gives recommendations in case of usage questions. The CM group advises on such matters as the component structure of the release that the project is working on. The CM group helps to define a hierarchy that best fits the requirements of the projects

CMVC is also being used as part of the management instruments. Another group has been put in place to gather metrics data from CMVC and use it to get reports and charts about the status of the various projects. At the time of writing, the data is pulled out of CMVC and fed into a database system running on PC/DOS, where existing presentation procedures centered around PC tools could be reused. Eventually similar reports and graphs can be produced directly from CMVC as explained in 4.7.1, "Back-End Tools" on page 61.

What is important to realize is that CMVC is not just used as a tool within the development effort. It has evolved to become part of the company's entire development approach including such areas as design, test, and management. As an example, MCI also uses CMVC for nonsoftware development and stores course material for education as well as process descriptions.

The Programming Center at IBM Tucson develops the ADSM product, which in 1992 also began to address distribution aspects and was being developed on

workstation and PC platforms. At that time, the host-based development used a host-based CM library with limited capabilities. For example, remote platforms were not supported properly, and there were no clients available in the workstation environments.

The Programming Center started to look at PVCS from Intersolv and used that product for a while. However, it soon discovered that the lack of integrated problem tracking was a major deficiency compared to its requirements. It did an analysis and decided to migrate to CMVC as its CM tool for all environments, but in order to keep the history data of PVCS, which it had been using for almost a year, it decided to use PVCS as the underlying version control system. The access to the files, however, is only done through CMVC, so PVCS became almost invisible to the end users.

Since then, the development process has been well integrated with the functional capabilities of CMVC. As explained in 4.2.3, “Networking and Remote Access” on page 51 several other development locations use the CMVC server in Tucson from all over the world, and they have to use CMVC as part of their development process as well. The test groups use canned reports to get information about the test status and progress, and even groups that do not control their source code with CMVC use it for problem tracking such as the Information Development group, which writes the official product publication.

Two dedicated support people are working in the Programming Center for CMVC family administration and end-user support. They developed a play component within CMVC to be used to try out CMVC, and they also provide additional educational and presentation material to potential new end users.

Also, the tape and DASD drive developers at IBM Tucson earlier used other IBM internal CM tools. They also used a lot of reporting and graph-viewing facilities on their VM systems. A key factor for a successful introduction of CMVC into the process was the requirement to have a similar functionality and to provide similar output data as the tools that were used before provided.

Therefore, a specific support group was put in place that had to provide support for family and system administration. The group also had to provide the proper tools for test, development, and management and consultant advice for the various user groups. This consulting included help during the initial setup for the component hierarchy, authorization setup, and CMVC customization, such as for variable fields, choices, or user exits. The CM group also has developed education material that is used during classes.

4.3.3 Acceptance and Feedback

Whenever a new tool is introduced into an existing process there are acceptance problems at the beginnings. Developers sometimes tend to complain that using a tool for CM would cost them a lot of time, and the additional cost would not pay off compared to what is gained by the CM tool.

Although this seems to be obvious at first glance, it is not true because effort has to be spent in any case for configuration management even if it is done manually without a supporting tool. Almost all development shops not using an orderly CM tool have faced severe problems when they had to reconstruct and reproduce code releases.

SCS and MCI reported software failures of the software they had developed without a CM tool, and the costs of that failure would be immense. If the software that has to be maintained is part of the company's integral business, it is absolutely required that at any time any source code in production can be accessed, fixed, and maintained immediately. Developers, who have experienced this on their own, no longer doubt the requirement of using a CM product. They answer the question, “Does this extra effort pay off?” with a definite “Absolutely!”

CMVC does not only help during the development and build cycles of pure development. It is also a very important and useful tool for getting exact and up-to-date status information. Management is looking into the following data:

- How many defects are currently open?
- What is the evolution of defects over time?
- What is the distribution of defects regarding severity?
- What is the mean time between defects?

At MCI the answers to these questions are usually presented in graphical form, and the source data for these plots are gathered from CMVC. “Managers and metrics people love it immediately,” is one good example of the acceptance from these groups.

To get good feedback from other CMVC users such as testers or developers MCI chose to gradually introduce CMVC. At the beginning of the project, CMVC was used without binding control to have minimum impact on the development process. At later stages of the project, when independent test groups require a higher degree of configuration management, binding control is turned on and integrated problem tracking is used.

At IBM Tucson a key requirement for CMVC acceptance was to provide output charts and reports similar to those produced by the tools that were used so far. This problem is addressed by having report generation routines and graph plotting utilities that gather the data from CMVC using the CMVC Report command and then feeding the data into whatever output processing tool is used. See Figure 4 on page 61 for an example of an output analysis.

Another item that improved acceptance from the end users was CMVC's capability to offer a command-line interface. This allowed an interface to CMVC to be developed from within another front end that is used as an interface to software developers. By doing so, the most common CMVC actions are made available from another graphical user interface. See 4.7.2, “Front-End Tools” on page 62 for details.

4.4 Diversity of Data Controlled with CMVC

When using a product like CMVC in a software development environment the first type of data one may consider to put under library control is source code. In fact, all of the sites that we visited use CMVC to control source code. However, other types of data can be stored under CMVC control to take advantage of either the configuration management functions or version control features of CMVC.

4.4.1 Source Code

Source code controlled from a CMVC server on AIX means C source code at the first glance. This is mostly true, however, only on AIX or similar UNIX derivatives. Other programming languages such as shell scripts are controlled by IBM Tucson's DASD development groups. They also store Assembler code using CMVC, and MCI stores C++ code using CMVC.

The tool support group at IBM Tucson has developed a simulator tool that allows animation of the design of models built using the CADRE TeamWork OOA product. This simulator is written in SmallTalk, and the code for this simulator is also maintained using CMVC.

To separate the code that is maintained for development support from the code that is developed to be shipped to customers later on, IBM Tucson chose to use a different family and put the internal tool code into that family.

Source code does not only include program code. For example, other source code may be an AIX message source file that needs to be processed with the `gencat` command in order to be accessible at run time from another program. Another example of source code may be a series of X11 resource specifications, which during build need to be combined with one X11 resource file that is used for the application that is to be built.

4.4.2 Build Tools

Most of the sites that we visited also use CMVC to store those source files that are related to the build process. These build tools are specific for the different target platforms on which they run. For example, in the case of AIX or other UNIX platforms, the build files stored and maintained with CMVC are make files. In the case of other operating systems the build tools consist of programs that theoretically can be written in any language.

For example, the Programming Center at IBM Tucson uses REXX EXECs to compile, link, and build the ADSM product deliverable for the VM operating system. This REXX source code is controlled by the CMVC server on AIX, and it is extracted during a release extraction like any other piece of code that is required for that release. Once the extraction is finished, the build EXEC is started on the VM target platform.

A similar approach is taken for the other target platforms such as MVS or OS/2. Here, the build programs would consist of CLISTs (MVS) or command files written in REXX (OS/2).

The build programs mostly deal with compile and link commands, but, depending on the application and its complexity, other build steps may be required. For instance the build procedure also has to build the proper X11 resource file for a Motif application, and message catalog files if the application uses the message handling facilities of AIX.

4.4.3 Test Cases and Test Tools

Test cases and test tools are treated in a similar way as source code and build procedures. Although it may not appear required at first glance, there are a couple of reasons why the test cases and tools are also maintained using CMVC:

- Test cases are associated with a certain functionality of the code, and this functionality is controlled using the version control features of CMVC.
- Test tools may be used for automatic execution of tests and are also bound to a specific level of code.

It is then only consequent to use CMVC to maintain the test environment using CMVC.

Some projects choose not to store their test tools and test cases using CMVC. This may be the case if there are no test tools at all, or the test environment is totally separated from the CMVC server. However, even then the problem reporting and defect tracking functions of CMVC can be used to report program bugs against the components.

One project at IBM Tucson decided not to report the defects immediately against a component used to control source files. Rather than that, it defined a dummy component that is purely used for problem tracking purposes. The component owner is the test leader and test coordinator, who is a filter of all of the reported problems from the test group. This person decides whether each defect is valid to be passed on to the development groups or is an invalid or duplicate defect. He or she also decides on the severity of the defect and can modify the description or add more information that further helps the developers to fix the problem. Thus the workload of the development groups is reduced. The test leader also has a better knowledge about which source component is the correct one to which to route the defect.

A similar approach (dummy component) is used if the defects are reported against files that are not under CMVC control. Although this may sound strange at first, almost all of the sites visited used the CMVC problem tracking features for files which were not under CMVC version control. For example, some hardware development groups at IBM Tucson have a dummy component for hardware development. They do not use CMVC during the versioning of hardware design, as the engineering process of the hardware is somehow different from the software development process. However, they also use the CMVC problem tracking features.

4.4.4 Documentation

Documentation is another kind of data that is put under CMVC control. Documentation material includes:

- Process documentation, which describes how the company, site, or organization develops software. This description can be used to describe the quality processes required for getting ISO 9000 certification.
- Product and tool documentation, which describes any aspect of the controlled tool, project, or product. It covers administration, installation, user reference, and other similar topics.

IBM Tucson stores various types of product and tool documentation under CMVC control but not process documentation. MCI started using CMVC for process

documentation and wants to use CMVC to control the documentation about the MCI development methodology.

4.5 ISO9000 and CMVC

The ISO 9000 series of quality standards define a guideline for software quality assurance. Companies operating on a worldwide basis face the requirement of being certified as ISO 9000 compliant, as this certification becomes a prerequisite for a bid in many cases, especially in Europe.

CMVC can be used as a quality instrument during the development process to cover aspects such as document control, design control, product identification and traceability, inspection and test status, nonconformance and internal audits. It can also be used to store and control the documentation of the development process itself.

4.5.1 CMVC As Part of an ISO Certification Process

MCI is looking toward ISO with increasing interest due to the importance of ISO in the European market. It uses and views CMVC as one of the tools to move toward ISO certification.

IBM Tucson was certified in 1993, so it uses CMVC as part of its defined development processes.

Both locations use CMVC for *document control*, although this use is not yet consistent across all projects. They use CMVC partly for *design control*, but again the use is not yet a standard because some of the design is done on another platform using other techniques and tools (IBM Tucson), and sometimes the tools that are used for software design have a versioning concept of their own (MCI). However, both sites use the version control functions of CMVC during the software development phase, and this meets the ISO requirements of *product identification and traceability* as well as *inspection and test status*. In addition they use the problem tracking functions of CMVC, which meets the requirement of *control of nonconforming product* of ISO.

Results of internal quality audits are not yet reported against a CMVC component to record identified shortcomings of development processes. Instead IBM Tucson records these using another tool running on VM.

4.5.2 CMVC As a Repository for Process Documentation

None of the midrange sites visited uses CMVC to store the process documentation. MCI is in the process of adapting a methodology document about host-based development to the procedures and processes they use for workstation and PC development but has not yet completed that. IBM Tucson uses an existing VM-based repository for storing the process documentation. It did not use CMVC because everyone on the site has access to a VM system, but not yet to AIX, and immediate access to the quality documentation is one of the requirements of the ISO norm.

4.6 Organizational Consideration

Once the CMVC installation crosses a certain threshold a dedicated CM support is usually put in place to help and assist the increasing number of users and projects. This CM support is part of a general support group, which in addition can handle project management, test or system administration (such as with MCI). CM support can also cover tool development and support (such as with the Design and Test group at IBM Tucson). The CMVC administration tasks are mostly closely related to the general system administration, but sometimes they are also clearly separated, as in the Programming Center at IBM Tucson.

4.6.1 Central Support for Rollout

When a new project starts to use CMVC, or new users join a project that is already using CMVC, additional support is required to achieve a proper and smooth transition. The CM staff at MCI has developed education material on its own, which is particularly suited to the needs of MCI. This education material focuses on the roles of the people within the development process of MCI and has copied and modified the original material from the official IBM publication.

In addition, the CM staff has developed classes that address the situation of MCI in particular. These classes show how MCI has customized CMVC (for example, the modified task list) and explain how CMVC is used at the site during the development process. Project-specific customizations such as certain user exits or configurable fields are introduced during these classes, as well as topics that address MCI-specific procedures to support non-AIX platforms.

When a new project is started the CM group also acts as a CMVC family administrator. The release or family definitions and setup are discussed with the project leader, and the family administrator helps with the initial setup for the family, release, and component hierarchy. The family administrator also defines the users and their access authorities to the files under CMVC control.

4.6.2 CMVC Administration

In smaller development organizations the tasks of the CMVC administration are almost always part of the to-do list of the overall system administrator. However, larger shows with a higher degree of networking and a larger community may decide to separate the tasks of system administration and CMVC administration. In the case of the Programming Center at IBM Tucson this task split was also done because the group members handling the CMVC administration were not yet deep AIX system programming experts.

The day-to-day tasks of CMVC administration are:

- Define new families or releases
- Setup for new users
- Customize configurable fields, choices lists, and user exits
- Plan for backup.

A site just beginning to use CMVC usually just uses the canned CMVC customization and may even take some optional processes out. So at the beginning user support and initial setup are most important.

Over time, however, projects begin to discover the various ways in which CMVC can be customized to the particular needs of the project, and more requests for individual support arise.

4.6.3 User Support

At medium-scale sites the CMVC administrator mostly provides end-user support. Both MCI and IBM Tucson do not have a dedicated support person or help desk just for CMVC questions, as is the case for very large CMVC sites (see Chapter 5, “Use of CMVC on Very Large-Scale Basis” on page 75).

The administrator also supports the development groups with problems they may have during their build process and provides answers to end-user questions about how to use CMVC.

4.6.4 Tool Support

CMVC's open architecture is based on the availability of a command line interface and the underlying use of an RDBMS, which allows raw output data to be gathered. Thus, a lot of back-end and front-end tools can be developed around CMVC, and in medium-scale and large-scale sites this is often the case.

The use of the interfaces that CMVC provides usually requires a thorough understanding of possible side effects. So the people working on tool development need to have a deep knowledge of CMVC or at least work together closely with someone who is quite familiar with CMVC's interfaces.

At the study locations representing medium-scale CMVC sites the CMVC administrator is the focal point for tool development. See 4.7, “Extensions to CMVC” for examples of CMVC customizations at MCI and IBM Tucson.

4.6.5 System Administration

When a company or location decides to use CMVC, usually the system administrator is the first person to take over tasks that are related to CMVC administration. However, in some cases, these two roles are split, and the jobs are assigned to different people.

This is the case if system administration functions are available from another (central) support organization. Some tasks, however, address both the CMVC administrator and the system administrator:

- Planning for backup and restore
- Migration to new version and release levels of CMVC
- Installation of PTFs.

4.7 Extensions to CMVC

This section describes the locally developed front-end and back-end tools that are used to exchange data with CMVC. It also describes the impact of these tools on the overall development process and the relation of these tools to the use of CMVC.

4.7.1 Back-End Tools

Back-end tools are procedures put in place to get data out of CMVC for further processing. A development location could use the CMVC GUI to look at the various reports, but typically the output data is processed further on to get more appealing charts or graphs.

4.7.1.1 Graph Plotting

All of the development locations that we visited during this project are trying to generate graphical representation for management information from the data out of CMVC. Some of the sites have developed very elegant and automated procedures to do this and have spent a lot of effort in getting the requested graphs. Others use the CMVC report functions to get the raw data and then manually reenter a subset of the output data into a postprocessing tool of their choice.

IBM Tucson uses existing tools written for a database based on VM/AS (Virtual Machine/Application System) to create the graphs. VM/AS is a project management product available on the VM operating system. A background procedure gets the data from CMVC and loads it into the database located on the host for later use by VM/AS.

Figure 4 shows a sample chart that is produced in this way. It shows the evolution of reported defects over time in a monthly analysis chart.

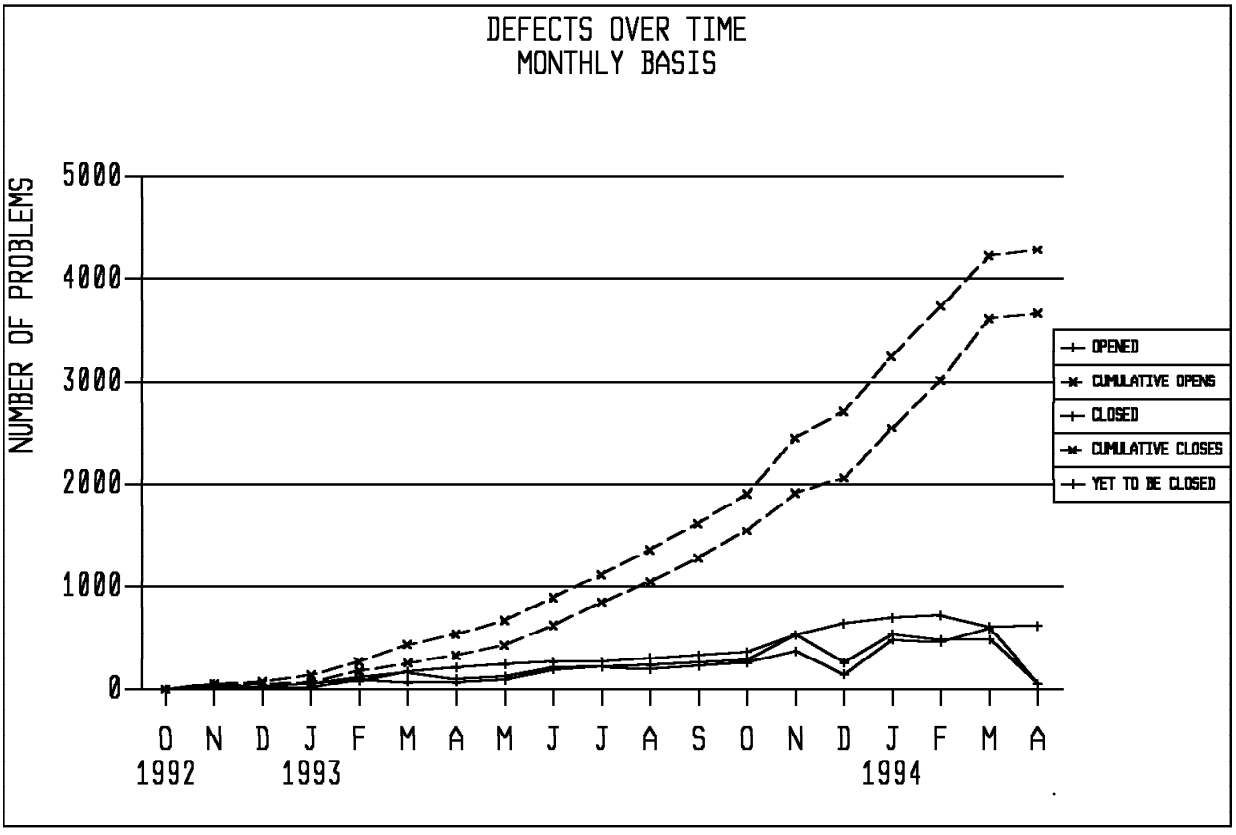


Figure 4. Sample Graph Showing Defects over Time (Monthly)

The graph shows the number of defects that are reported over time for a particular project. It shows the total number, the number of the closed defects, and the

number of those defects that are still to be fixed. This product allows report data and various graphical representations of the data to be produced.

Instead of using VM/AS, one could also use a PC spreadsheet program to illustrate the data in a graphical way. IBM Tucson has an automated procedure to automatically gather the data at night or on request from CMVC using calls to the command line interface of CMVC. The raw result data is then fed into the proper tool for the proper postprocessing.

MCI has not yet automated this step; it extracts the data using the CMVC report facilities. Then the data is manually transferred to a PC spreadsheet program, which shows a similar representation. The charts MCI produces show:

- The mean time between defects
- The test status with the number of cumulative, fixed, and closed defects
- The severity distribution of defect reports.

MCI is looking at ways to generate these charts automatically from the CMVC database.

4.7.1.2 Report Generation

Reports are another means of describing the status of a projects. Like charts and graphs they often serve as the base for management decisions and therefore are very important. Status estimations, risk assessments, and schedule outlooks are the most important management activities during a project, and the data (and the way the data is presented) is critical to most management decisions.

IBM Tucson had a report generation tool called PTS2 (Problem Tracking System 2) that was in use by various projects and that evolved as a standard throughout the organization. Therefore, when projects started to use the problem tracking functions of CMVC the requirement was raised to produce similar reports from CMVC as with PTS2 before.

To achieve this, the support group at IBM Tucson wrote an interface program that would extract the problem tracking data from the defect and the track table out of the CMVC database. The program then loads the data into the SQL/DS data on VM, where the existing utilities can be used to produce various kind of reports.

The reports that can be produced are:

- Bingo tables to show defect severity, problem area, wait code, and others by status
- Summary tables to show the number of open and closed defects.

The tool used to produce the reports is an interactive VM application, which also allows the mountain charts or pie charts to be generated to get a graphical representation of the same data.

4.7.2 Front-End Tools

In addition to tools that gather data from CMVC and further pass it on to other tools for postprocessing, there are tools that are used get data from other sources and feed them into CMVC. This section shows several tools that are in use at the IBM Tucson site. They run on various platforms and use the noninteractive command-line interface of CMVC.

4.7.2.1 ZAPAR

ZAPAR (Zaepfel APAR tool) is used to build a bridge between the Remote Technical Assistance Information Network (RETAIN) system and CMVC.[1] The IBM technical support groups use RETAIN to report Problem Management Records (PMRs) from customers to the development and maintenance organization. If a PMR turns out to be a valid program error, the PMR becomes an Authorized Program Analysis Report (APAR) and is treated as a defect against the corresponding software product.

The Programming Center in Tucson maintains the ADSM product and uses CMVC to control the ADSM source code. So whenever an APAR is received through RETAIN, a corresponding defect must be opened in CMVC against the corresponding release. This interface is automated through ZAPAR. Figure 5 explains the flow of data between RETAIN and CMVC.

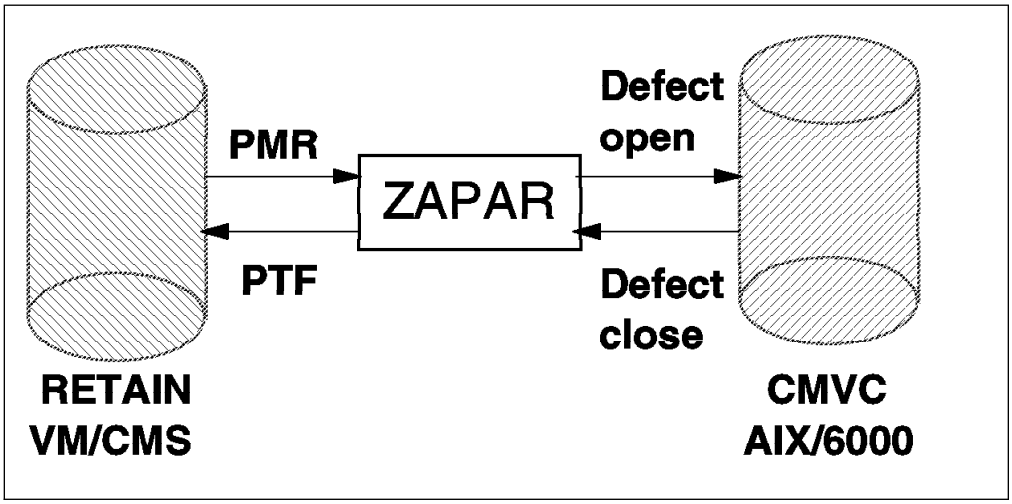


Figure 5. ZAPAR Interface between RETAIN and CMVC

The RETAIN system is accessed from VM, and so is ZAPAR. In addition, a ZAPAR version is available that runs on OS/2. The user interface of ZAPAR consists of a couple of screens with a few options. The functions that are available allow information to be obtained from RETAIN and automatically create a CMVC defect with the same data. Likewise when the user closes a CMVC defect from ZAPAR, the corresponding APAR is also closed in the RETAIN system. ZAPAR also offers functions that are purely working on CMVC, for instance, the user can modify the defect owner or the component to which the defect is assigned.

ZAPAR uses an IBM internal tool for its screen I/O, but the same functionality can be implemented using systems like ISPF or even XEDIT screens. ZAPAR uses the CMVC command line interface to get data from CMVC or to put data into it, and it uses a similar interface to RETAIN.

[1] Andy Zaepfel is the developer of ZAPAR

4.7.2.2 BuildTool

Another example of a front-end tool interfacing with CMVC is also in use at IBM Tucson. This tool (called BuildTool) is used to combine access to common development functions like edit, compile, and touch with functions to access CMVC like check in, check out, or extract. The name *BuildTool* is unfortunately misleading, as the purpose of the tool is not only to provide functionality during the build steps of a project or program. A more appropriate name to summarize the functionality would be for, example, *DevelopmentTool*.

The BuildTool is an interactive Motif application written in Korn shell and perl, an interpreter language that is available as public domain software. The size of the source code for the entire tool is about 2000 lines of code.

The user interface was designed using EZWindows, an IBM internal use package that offers rapid GUI prototyping functionality similar to AIX windows Interface Composer (AIC).

When users click on a push button or one of the items in the pull-down menus, they invoke proper callbacks. These callbacks then either call the corresponding AIX commands, such as `touch`, or call CMVC through the CMVC command line interfaces.

This additional front end interfaces with the command line interface of CMVC. It is a good example of the flexibility of CMVC and its capabilities to be integrated in another tool or process.

Figure 6 on page 65 shows the user interface of the BuildTool in use at IBM Tucson.

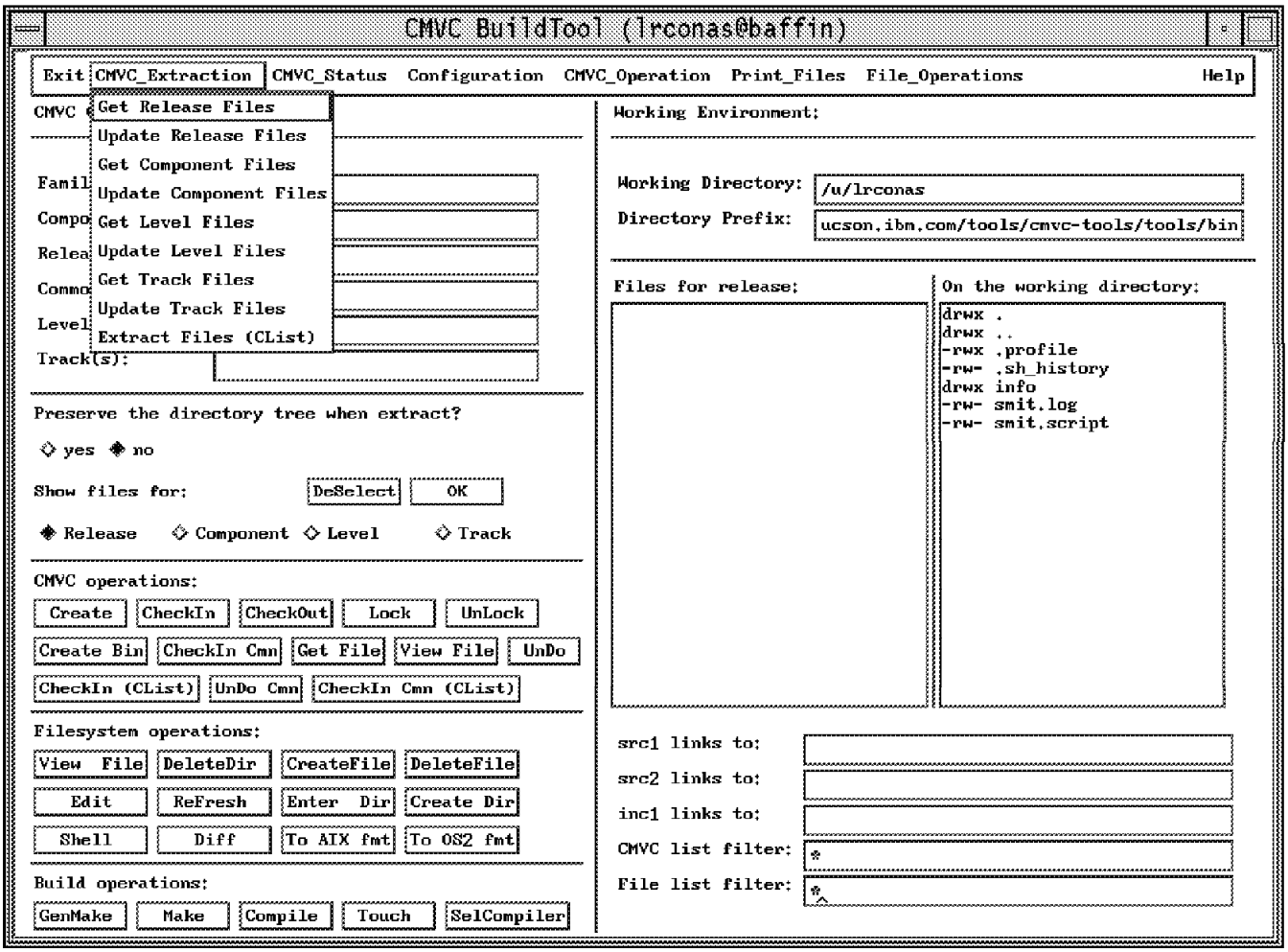


Figure 6. BuildTool : A Tool Interfacing with CMVC

The BuildTool offers both common development functions, such as:

- Actions against files (for example, browse, edit, create, delete, diff)
- Build activities (for example, compile, touch, make)

and CMVC functions, such as:

- Activities against selected files (such as check out, check in, lock, extract)
- Activities to define the scope of other CMVC actions (such as set the family, release, component)

4.7.2.3 Shared Working Space Manager

Another example of a tool that interfaces with CMVC is in use at the IBM Tucson site: the Shared Working Space Manager (SWSM).

The SWSM is a collection of demon processes that update a set of files with the most recent version from CMVC. A configuration file defines a shared working space as a target directory structure, which is a directory structure shared by multiple developers. The SWSM demons support the automated update of the shared working space. This update can be triggered periodically, for instance, through a crontab entry.

Thus with SWSM a set of files under CMVC control is shadowed to a target directory, so developers who need to share code always have access to the most recent code versions of their colleagues.

SWSM also can be used together with the BuildTool. The interface between the BuildTool and SWSM is enabled by setting an environment variable in the user's `.profile` file. Then whenever a user uses a function from the BuildTool that modifies the files in CMVC (such as a Checkin, Delete, or Undo), the BuildTool automatically informs the SWSM by a message exchanged through TCP/IP.

The SWSM demon then extracts the current version of the file and stores it in the working directories that are specified in the customization environment of the SWSM.

You can achieve function similar to that provided by the SWSM by using the various CMVC user exits.

4.8 Customizing CMVC

Development locations of a size such as MCI or IBM Tucson usually have customization needs beyond the CMVC default customization. Areas of CMVC customization are:

- User exits to perform specific follow-on processing for the corresponding CMVC actions
- Configurable fields in the different tables to store and process additional data
- Customized choices to meet project-specific or companywide standards
- Configurable processes to tune the CMVC processes to the approach most suitable for the project needs.

4.8.1 User Exits

MCI has one user exit implemented in a production environment to solve an interesting problem for a non-IBM platform (Stratus), which allows a different syntax for file names than the one used on AIX. The file names must be mapped between the Stratus and the IBM platform before they are exchanged. Special characters in a file name, such as a dollar character, which is valid for the Stratus/VOS operating system but invalid for AIX, need to be mapped to characters that are valid for AIX. MCI implemented an elegant way of doing this by providing a user exit, which is called for Checkin, Checkout, and Release Extract operations. For these CMVC transactions the user exit maps invalid AIX characters (such as the dollar character) to other valid special characters (such as an underscore).

MCI also thinks about invoking another user exit upon a File Checkin to produce a 'pretty printed' listing of the source files, which then can be centrally archived for access from any other developer.

At the time of writing the Programming Center at IBM Tucson was not using user exits. It plans to put one in place to provide an automatic extraction of files to a common target directory after the file is checked in. This function is similar to that currently provided by the SWSM (see 4.7.2.3, “Shared Working Space Manager” on page 65).

Another user exit under consideration would automatically create tracks for multiple releases, if a defect is created. Right now, this is a manual activity.

4.8.2 Configurable Fields

MCI uses a configurable field to indicate a path name for a file. This path name is used as a prefix pathname if the file is going to be extracted to a remote Stratus platform.

At IBM Tucson there are not yet additional fields configured. However the following additional fields for the defect table are under consideration:

Impact code Identifies what is the impact for the end user if this defect is encountered. This field is used in a similar way as the *Severity* field, but *Severity* is used to describe the impact for the test rather than the end user.

Probability Identifies what the probability is to encounter this defect.

Phase injected In addition to the *Phase found* field, which specifies the development phase where the defect is detected, this field specifies the development phase where the defect is injected into the product. The contents of the field is used for defect prevention activities. For example, if a particular phase shows up to inject many errors, then the development process for that phase needs to be corrected.

4.8.3 Choices

All the sites that were visited have adjusted the list of choices that are available for specific fields. Some of the choices for a field are taken out, others are added. The range of allowed values can be adjusted to match the particular development process and the terminology that is used throughout that location.

4.8.4 Configurable Processes

One of features of CMVC is the possibility to dynamically configure the CMVC processes, and mid-range locations such as MCI and IBM Tucson use this feature a lot. The introduction of a product such as CMVC can be made much smoother, if for example binding control is turned off at the beginning of development. Later, when a separate test group is joining the development process and configuration management becomes more important, binding control is turned on and more of the functionality of CMVC is being used.

At MCI for instance most projects do not use approval records, and test records are used only for a small number of projects. Thus a more controlled process may be introduced if the project complexity and its organization requires it, and CMVC is flexible in handling this.

4.9 CMVC Used with Different Development Paradigms

This section describes the use of CMVC in development environments that choose a different development paradigm from the standard waterfall development model with procedural programming languages. Groups at IBM Tucson work with OOA methods during their analysis and design phase, and they use CMVC in combination with other tools such as Cadre TeamWork and a Smalltalk development environment. They also make a first step toward cleanroom development by following a hybrid development methodology.

4.9.1 Object-Oriented Analysis

The DASD and tape drive development groups at IBM Tucson use OOA methods to model the behavior of the software system to be built. The goal of the model is to first get a clear understanding of the underlying problem, but furthermore tools are put in place that allow the model to be simulated and thus errors to be detected very early in the development phase. On completion of the OOA methodology, the design will have fewer errors and be of better quality, and thus overall development cycle time is reduced.

OOA according to the Shlaer-Mellor definition leads to the following:

- A data view, which is documented as an Entity/Relationship (E/R) data model
- A control view, showing the different states an object may have
- A processing view, which may be expressed by state transition diagrams.

As a result, OOA models the characteristics of the data along with the dynamic behavior of the data objects, when a transition from one state to another occurs.

OOA does not specify a particular syntax for defining the state transitions. However, any automated verification of an OOA model is based on a formal grammar, so IBM Tucson has defined a syntax called Process Specification Language (PSL). This syntax allows dynamic state transitions to be modeled using a formal grammar.

4.9.2 Verification of the OOA Model

IBM Tucson has developed a simulator called the Tucson OOA Simulator (TOOAS), which can be used for verifying the dynamic aspects of an OOA model.

IBM Tucson uses the Cadre TeamWork product as a front end to develop the OOA model. Using TeamWork, developers model the dynamic behavior of the objects with PSL.

Once that is done, the OOA models are extracted and the PSL behavior specifications are compiled using a PSL compiler. The PSL compiler checks the PSL specifications for syntax errors and semantic errors and then generates Smalltalk code. The TOOAS then uses the Smalltalk code to interpret the model behavior and offers a dynamic view of the OOA model. TOOAS allows developers to observe the state transitions of the modeled objects to see whether they behave correctly. The TOOAS user interface allows developers to change the state of objects or send other external events to the model being simulated.

If errors are detected, the designer uses TeamWork again to modify the model and then runs the extraction and PSL compilation again to simulate the corrected model.

4.9.3 Use of CMVC Together with OOA

As explained in 4.9.2, “Verification of the OOA Model” the DASD and tape drive development groups in IBM Tucson use Cadre TeamWork during their design phase along with the PSL compiler and TOOAS. At some point the models need to be implemented using programming languages such as C or C++. The different projects at IBM Tucson choose different languages.

At this point of the development process so-called application objects are developed, each consisting of a set of files that implement each object's behavior. The application object code implements the attributes, states, and events for each object.

One project uses another generator tool to ensure that the design of the object always matches the application code. To do this, header files are automatically generated from the extracted model. These header files, for example, define the names of the states that an object can have, and these definitions are also used in the manually implemented application object code. Thus whenever the application object is changed, the corresponding model also needs to be changed and the header files need to be regenerated.

The component structure set up in CMVC to develop the application object corresponds to the E/R models and objects as design in Teamwork. For each release in CMVC the associated model in TeamWork exists with a different name. A naming convention is defined such that for a given release the corresponding release of the TeamWork model can be retrieved.

Another project at IBM Tucson does not keep the OOA models up to date. Rather it uses TeamWork during the design phase only and then moves to CMVC to control the implemented application objects. If a defect is found that would require a change to the OOA model, this is no longer reflected in TeamWork. The project freezes the design documentation at a specific point of time in the development process.

4.9.4 Cleanroom and Parallel Development

IBM Tucson also tries to follow the paradigm of cleanroom development. In theory cleanroom development defines a development process where linear increments are defined for a given project. Each increment can be viewed as a small functional piece of the overall product. An increment is fully developed, tested, and corrected before the next increment is begun.

The team using the principles of cleanroom development is developing the disk and tape drives and does not follow the methodology completely. Instead the team implements and tests several increments at the same time, which requires a much more complex release structure within CMVC. The reason for this modified approach (hybrid development process) is the schedule and availability of the underlying hardware. In addition, the tests for the different increments have to follow one another immediately, which forces the development group to work on multiple increments simultaneously.

The parallel development of multiple increments is reflected in CMVC by having multiple releases under development at the same time. A source file is under development in multiple increments, typically three at the same time. When development starts a new increment, a new release is created in CMVC, and all files from the previous release are linked to the new release. The common links are broken, and thus a new independent code level is created for the new increment.

However, this decision has some impact on the way defects are implemented. When a defect is reported during the test of an increment, tracks need to be opened for each release that is currently under development for that file. As there is no more common code across releases, a fix for an early increment must be

manually repeated for each follow-on increment that is also under development. This makes a lot of additional work for the development teams, but it is the only way to work on multiple increments at the same time.

It is obvious that in this parallel development effort CMVC plays a key role in ensuring a correct configuration management and version control. As a team leader at IBM Tucson stated “CMVC has held us together.”

4.10 Firmware development

One group at IBM Tucson is developing system embedded software (microcode) for the control units used for disk drives. This firmware development group also uses CMVC for configuration management purposes.

During the design phase this group uses the Cadre TeamWork product to document the design. As a result, the design is modeled in various ways, such as E/R data models, data flow diagrams, and state transition diagrams.

The group starts with the actual coding phase at a certain point in the development process and uses CMVC from that point on. It uses CMVC to maintain all code that is developed and the GNU C++ compiler to compile the code.

The problem the group faces during the test is that the hardware onto which the final compiled code is loaded cannot be used for the test because there are no test tools that actually access the new hardware. So the group uses the C++ compiler as a cross-compiler to generate code that matches the underlying hardware. A linker links together all of the different object modules that are part of the firmware. Then a simulator tool running on AIX loads the object modules and runs specific test cases.

The simulator offers a command language interface so that more complicated test cases can be run, and it also has an embedded debugging facility to ease problem determination.

Once defects are found, they are reported against the corresponding component in CMVC. The developers fix the defects as for any software development project, and the corrected new level of code is extracted during the next release extract. The group uses a utility to generate make files once the code that is to be built is extracted, rather than storing and maintaining make files manually. This "make make file" utility generates a new make file that reflects the dependencies of the current (extracted) level of code. The make file is then run, the sources are cross-compiled, and the object modules are then loaded into the simulator.

At a certain point in the development cycle the test would be run on the real hardware. As before, the extract is run from CMVC, the make file is generated, and the sources are cross-compiled. Instead of loading the object modules into the simulator, they are written to floppy disks, which are loaded into the real hardware. Then the hardware and firmware can be tested together in an integrated environment.

4.11 Data Base Considerations

CMVC does not use a proprietary database; it uses a relational database to store the control information. Various database systems can run underneath CMVC, and different backup and archive strategies can be applied. This section discusses one example of a backup strategy, as it is used by the Programming Center at IBM Tucson, to demonstrate that backup can be completely automated.

4.11.1 Choice of Database

CMVC supports multiple RDBMSs from different vendors, such as Oracle, INFORMIX, Sybase, and DB2/6000. There can be multiple reasons why to choose a particular database system; for example, database system may already be in use at the customer site for other production systems. Price/performance ratio is another item to look at, and several sites that were interviewed during this red book effort in fact found this more important than the fact of introducing another RDBMS.

However, there is a particular impact on the overall administration when a database system is introduced. Regular tuning and backup procedures must be put into place, and the time and effort required for initial setup and customization must not be underestimated.

4.11.2 Backup and Restore

When CMVC is used in a production environment, a strategy for backup and restore must be defined and set in place. There is no right or wrong strategy per se, but the general goal is to archive data periodically so that in case of a severe problem the loss of data is minimal (if at all).

A typical backup strategy for a CMVC installation on top of Oracle would consist of the following steps:

- Kill each family's CMVC daemons.
- Shut down and start up the Oracle database to empty all buffers and clean up memory.
- Export all database tables to a temporary hard disk file.
- Unmount all NFS-mounted directories so that the tar backups do not copy their information.
- Use the `tar` command to create a backup of the file systems found in `/cmvc.families`.
- Reactivate each family's cmvcd and notifyd daemons.

Below we show how the Programming Center at IBM Tucson uses ADSM to back up CMVC data. It is interesting to see the various roles of clients and servers in this backup strategy: ADSM is run as a client on the CMVC server system to request the backup of that system, and an ADSM server on a connected host is used to back up the data onto a VM system. The system center of that VM system performs regular backups of that data, so that if a restore is required the data can be reclaimed from VM back to the CMVC server machine.

The typical execution time for the regular backup job is less than 10 minutes, based on the IBM Tucson site characteristics as described in 4.1.3, “IBM Tucson” on page 47.

Figure 7 on page 72 shows the crontab entry that is used to perform the backup of the CMVC installation at IBM Tucson. The crontab entry consists of a call to two

separate shell procedures. One shell script (*dsave.sh*) is used to bring down CMVC, back up the datebase, and restart CMVC again. A second shell script (*cmvclog.sh*) then cleans the CMVC log files.

```
0 0 * * * /home/mmahoney/dsave.sh
10 0 * * * /home/mmahoney/cmvclog.sh
```

Figure 7. Crontab Entry for Backing Up the CMVC Server

Figure 8 shows the shell script that is used to back up the CMVC server.

```
#!/bin/sh
echo stopping the CMVC daemons >>/tmp/cron.log
/usr/lpp/cmvc/samples/stopCMVC df >>/tmp/cron.log
#
# to restore the database being exported enter:
# /usr/informix/bin/dbimport -ansi -c -l -i /usr/informix/exportdf df@DFONLINE
# while logged in as informix
#
cd /usr/informix
rm -R /usr/informix/exportdf
mkdir exportdf
chown informix.informix exportdf
echo exporting the database >>/tmp/cron.log
su - informix "-c /usr/informix/bin/dbexport -o /usr/informix/exportdf
df@DFONLINE 2>export.err"
echo leaving exportdb >> /tmp/cron.log
#
# Using ADSM to backup filesystems - for system and CMVC data
#
DSM_CONFIG=/usr/lpp/adsm/bin/dsm.opt
DSM_LOG=/tmp
export DSM_CONFIG DSM_LOG
echo ------------------------------>>/tmp/cron.log
echo ADSM run at  date  >> /tmp/cron.log
/usr/bin/dsmc incremental >> /tmp/cron.log 2>&1
echo ==============================>>/tmp/cron.log
#
# Restart CMVC daemons
#
echo entering startcmvc shell >> /tmp/cron.log
su - df "-c /usr/lpp/cmvc/bin/cmvcd df 9 &"
echo starting the notify daemon >> /tmp/cron.log
su - df "-c /usr/lpp/cmvc/bin/notifyd &"
echo ending daily backup script for  date  >> /tmp/cron.log
echo ==============================>>/tmp/cron.log
#
exit
```

Figure 8. Shell Script (dsave.sh) to Back Up the CMVC Server

Figure 9 on page 73 shows the shell script to call another shell script to clean the CMVC log after the server has been backed up.

```
#!/bin/ksh
#
su - df "-c /home/mmahoney/cmvclog.clean"
exit
```

Figure 9. Shell Script (cmvclog.clean) to Start to Clean the CMVC Log

Figure 10 shows the actual shell script that is run to perform the log cleanup.

```
#!/bin/ksh
#
#
if   -d $HOME/audit “
   then
   cd $HOME/audit
   mv log.6.Z log.7.Z  2> /dev/null   # slide the online.log files down
   mv log.5.Z log.6.Z  2> /dev/null
   mv log.4.Z log.5.Z  2> /dev/null
   mv log.3.Z log.4.Z  2> /dev/null
   mv log.2.Z log.3.Z  2> /dev/null
   mv log.1.Z log.2.Z  2> /dev/null
   mv log.Z   log.1.Z  2> /dev/null
   compress log        2> /dev/null   # compress the online.log file
                                      # output is online.log.Z
   touch log           2> /dev/null   # create new empty online.log
   chmod 600 log       2> /dev/null   # set access -rw-------
fi
exit
```

Figure 10. Shell Script (cmvclog.clean) to Clean CMVC Log

4.12 Build Process Considerations

At the same time that a development project is divided into pieces that are assigned to different developers, some thoughts must be given to the build procedures that build the entire application. The complexity of these build procedures does not necessarily correlate with the size of a project, the size of a site or location, or the number of individual source files.

For example, a huge project with dozens of developers may develop a lot of C source files, but the final build would only consist of a global compile and one link command. However, a much smaller application may require additional build steps:

- The team may be using a tool that requires a certain generation step before actual source code is generated. For example, the Application Development Lab in Hannover uses AIC to design the GUI. During the build steps source code reflecting the AIC design must be generated using AIC code generation utilities.
- The application may use AIX message handling routines, so the build procedures have to generate message catalogs from message source files that are used and accessed from the application at run time.
- The build for a remote platform may require some mapping to the target platform. For example, MCI uses mapping for file names because the syntax for a valid file name differs between AIX and Stratos/VOS.

- The build may also include some cross-compile if the target platform is different from the development environment.

It is advisable to store all source code including build tool source together with the application source code under CMVC. Once a CMVC release is extracted to a target directory structure, the corresponding build programs and their associated customization files (if any) are also extracted. Then the build procedures can be run on the extracted sources. This approach ensures that the build procedures always match the extracted sources. Errors during build can be reported against the CMVC components that are used to store the build programs and procedures.

Most sites have a specific CMVC component that is used to contain all files related to the build procedures. Sometimes it is even possible to generate the make files automatically after the source files have been extracted. The DASD development group at IBM Tucson uses this technique. After the source files are extracted to a target directory, a make file is generated using a `mkmf` command. This command scans all of the source files and automatically generates a `makefile` with a correct dependency list of the corresponding include file structures.

Some development locations, such as the Software Development Lab in Hannover, uses the SDE WorkBench as the development environment. The Hannover Lab uses the Program Builder component of SDE to generate the initial make file and then update dependencies if they have changed. The template for the make file, which is used by the `mkmf` command, needs to be adjusted and customized if specific rules or dependencies need to be taken into account. This is the case if for instance message catalogs or X11 resource files need to be created during the build process.

If the target platform is not AIX or UNIX, a tool that automatically generates the build procedure may not be available. In such cases, the build procedure needs to be stored like source code on the CMVC server, although the build procedure would never be run there. For example, the Programming Center at IBM Tucson uses REXX EXECs to compile, link, and build the ADSM product deliverable for the VM operating system. This REXX source code is controlled by the CMVC server on AIX, and it is extracted during a release extraction like any other piece of code that is required for that release. Once the extraction is finished, the build EXEC is started on the VM target platform.

Chapter 5. Use of CMVC on Very Large-Scale Basis

This chapter describes the use of CMVC at IBM Austin, a site representative of the world's largest CMVC community user. IBM Austin uses CMVC to develop and maintain the AIX operating system and other program products available for the AIX and OS/2 operating systems. This chapter shows how CMVC is used on a very large scale and has become an absolutely mission critical and integral part of the overall development process. IBM Austin uses CMVC in a highly customized way and has built multiple tools around CMVC that further process data gathered from CMVC.

5.1 Representative Customer

Moving from home-made CM tools to CMVC, IBM Austin demonstrates an impressive use of the product, with staggering numbers as far as the scale is concerned.

5.1.1 Evolution of CMVC

As described in 1.1, “Evolution of CMVC” on page 1 CMVC originated at IBM Austin, where a lot of experience in the area of configuration management led to the development of various tools, all of which were for internal use only. It soon became apparent, however, that these tools were required not only to plug a particular hole in the IBM Austin development process but also to solve a general software development problem.

After CMVC and its predecessors were used at IBM internally for a couple of years, the tool was also made available to customers and other IBM locations. IBM Austin uses CMVC in a highly sophisticated way, and the sections that follow discuss many of the unique features of its CMVC installation. IBM Austin uses CMVC to control various kind of data such as design documentation, specification documentation, source code, test case descriptions, test tools, build utilities, and process documentation. It uses Oracle as the underlying database system and plain SCCS as the version control tool.

5.1.2 Mission

The main mission of the IBM Austin lab is to develop the AIX operating system with all of the associated system software, including the Base Operating System (BOS) kernal software products such as Distributed Computing Environment (DCE) or similar software close to the operating system itself. In addition, the IBM Austin lab develops OS/2 products such as DCE for OS/2, LAN Requester, the High Performance File System (HPFS), and other software products in the area of networking and distribution. Most of this development effort is done in cooperation with the IBM lab in Boca Raton, Florida, which is in charge of developing the OS/2 operating system itself.

In total more than 2,000 IBM employees work at the IBM Austin site. This number includes the personnel for the hardware and firmware development groups and the corresponding production plants. In addition to the groups onsite, numerous IBM locations all over the world actively cooperate with IBM Austin during development, including groups in Europe and Japan for the National Language Support (NLS) of the program product. Also, vendors and other software contributors in the United

States for instance, the Open Software Foundation (OSF), also work closely together on the same software project. See 5.2.2, “Scale of Usage” on page 77 and 5.2.3, “Networking” on page 79 for more details.

IBM Austin also uses CMVC for problem tracking for the RISC System/6000 hardware and for problem tracking and release management of the IBM publication and documentation that goes along with AIX. The application software development groups developing application program products such as DCE also use CMVC during their development process. In addition to the AIX groups, several OS/2 development projects also use CMVC for various purposes.

IBM Austin is representative of a very large scale user of CMVC. This one lab is both a system software and an application software lab, a hardware vendor, and a nonsoftware development shop. The consequences for the site would be disastrous if CMVC did not working properly around the clock from both a performance and a functional point of view.

5.2 Usage Characteristics

Over the last years CMVC has evolved to become an integral and mission critical part of IBM Austin's development process The size and numbers of users, files, and other data stored under CMVC control along with the networking environment are huge and impressive. They show that CMVC is used by a site at the very high end of the usage scale.

5.2.1 Mission Criticality

In many aspects IBM Austin has incorporated CMVC into its entire development and maintenance cycle. At first, home-grown CM tools were used to aid the development groups in developing the first version of AIX. Over time, these home-grown systems were developed further to meet the requirement of the development processes, and other groups, such as hardware developers and the OS/2 development projects turned to what later became CMVC.

The AIX development groups use CMVC to control all of the AIX operating system code that is out in the field for all versions, releases, and modification levels that are in maintenance. Accurate configuration management is an absolute mission-critical requirement for reacting in case of defects found by external customers. None of the code out in the field can be reproduced without the help of CMVC, and the CMVC database is the only repository to keep track of the ongoing development and maintenance activities.

Future releases and versions are also being developed starting using CMVC. No other CM tool or backup system is maintained in parallel to CMVC, so the entire product development is based on this product. In total the development groups using CMVC including external vendors and other IBM internal development locations in other cities and countries sums up to several thousands. This huge number of people working concurrently with CMVC puts a top priority emphasis on the support and administration of CMVC.

A failure of the CMVC repository caused by a hardware failure, a problem in the network environment, or in CMVC itself would imply an enormous loss of productivity for almost the entire site. Such a failure of CMVC would cost several

million dollars per day, not counting the indirect costs incurred for other dependent locations, organizations, or customers.

The primary goal of the entire CMVC setup, support, and administration therefore is to ensure 100% availability around the clock. Automated procedures and tools have to be put in place to further support users, administer them, postprocess data out of CMVC, and customize CMVC to the needs of the various development groups.

5.2.2 Scale of Usage

IBM Austin is at the very high end of the CMVC usage scale. This is true for almost all characteristics of the environment.

5.2.2.1 Hardware and Software Environment

To split the work load among the different development groups and to split the various administration and configuration tasks, at IBM Austin the CMVC production installation for the various projects is separated into eight production servers. These servers are high-end RISC System/6000 machines like the RISC System/6000 models 980 and 950, and they are equipped with a main memory of about 256MB to 768MB. These servers run nothing else but CMVC.

As the environment is used around the clock, there is almost no time to upgrade to a new CMVC release or version online. Extensive preparation is required, and four other test machines are used to test modified CMVC code levels.

The total DASD capacity of the seven CMVC servers at the time of writing is close to 100GB of disk storage, and about 7,000 active users access these seven servers. The largest family is the one used to control the development and maintenance of AIX itself; there are more than 3,000 active users for that family. The server has about 1GB of data stored in the Oracle database, and about 15GB in the vctree. The Oracle table space is spread out over multiple RISC System/6000 drives to optimize disk access time.

The users accessing these CMVC servers are distributed all over the world and access the CMVC repository through local or wide area networks. See :figrefrefid=anetfa1 for an overview of the network topology.

5.2.2.2 Structure of CMVC Repository

Eight servers access about 30 families. The biggest servers still run at CMVC version 1.1, but the upgrade to version 2.2 will be done soon. Some other servers are already at that level, and whenever a new family is set up on a new server, that server will be configured to run CMVC 2.2.

The largest server maintaining the AIX operating system code controls more than 500,000 files, which are organized in about 1800 components. More than 3,000 active users work with this one family on this one single server.

The 1800 components are used for both configuration management and problem tracking. Some components are used only for problem tracking and release management, for example, those that represent the AIX system publication and documentation. This component controls about 150 IBM AIX books with about 20,000 associated files.

Once the development process reaches the function verification test phase, new build levels are built out of CMVC every 7 to 10 days. The compile, link, and extract jobs are run at night during low system load. In addition, service builds occur on a daily basis.

5.2.2.3 Statistics on CMVC Usage

Automated jobs collect data about CMVC usage at the site by looking at the CMVC audit log files. These statistics are then used to determine when batch jobs are started such as build procedures or other long-running transactions that produce output for reports, graphs or other postprocessing tools. Information about the system load and the characteristics of CMVC usage is required to fine-tune the system load. Long-running procedures that involve CMVC commands are therefore usually put into shell scripts, which are started through crontab entries.

Also, another set of crontab jobs monitors the CMVC installation and checks for potential problems such as a shortage of disk storage space. Automatic electronic mail is sent to both the system and the family administrators if critical situations are likely to be reached, so that proper corrective action (such as installing new disk drives, increasing the size of the file systems) can be taken before the error situation is actually encountered. IBM Austin wants to further automate this warning mechanism to automatically send an online paging message to the system administration staff.

The automated system that monitors CMVC usage showed a total average of more than 1.5 million CMVC transactions per day against the big CMVC family in April 1994. This family is used to maintain and develop the AIX operating system. This means that the server processes about 70,000 transactions per hour, and 20 CMVC transactions per second around the clock.

Table 3 shows which CMVC commands were used most during April 1994.

Table 3. CMVC Transaction Statistics: April 1994

CMVC Command	Frequency	Probability
Report	1,379,438	55.3 %
FileExtract	417,420	16.7 %
DefectView	258,531	10.4 %
TrackView	80,223	3.2 %
FileView	79,730	3.2 %
FileCheckin	25,807	1.0 %
UserView	24,926	1.0 %
FileCheckout	24,423	1.0 %
Others		8.2 %

The command processed by far the most is the Report command. This underlines the importance of CMVC in the overall development process at IBM Austin from a management perspective. In fact, CMVC has become the primary instrument and tool to gather all kinds of process control data that is required for status assessments and schedule outlooks.

5.2.3 Networking

Figure 11 on page 80 shows the various LANs at the IBM Austin site and the different bridges to other networks or dial-up lines. The seven large CMVC servers are in two different token-ring networks to have different physical access paths in case one of the rings encounters a hardware problem. The two rings are called the *Primary CMVC Ring* and *Secondary CMVC Ring*. Vendors and other non-IBM sites access the CMVC servers through another token-ring network called the *Isolated Vendor Ring*, which also allows remote dial-in connections through modems and connections through leased lines.

IBM developers from Austin are connected through several other token-ring networks running at either 4MB/sec or 16MB/sec.

In addition, the CMVC ring is connected to the host site network through PS/2 model 80 bridges.

The host site network also links together other IBM locations throughout the world. Thus remote development locations both inside the United States (such as Boca Raton, Kingston, Endicott, San Jose) and in other countries in Europe or Asia have direct access to the CMVC rings.

Another AIX machine acts as a mailer machine that creates raw data outputs for several families. It also generates a daily user's report for many users for many families. In addition, it acts as an NFS server for many of the off-site vendors. This machine exists in two domains:

- The IBM in-house network
- The vendor network.

By using NFS files like test cases or tar images can be passed between both domains and thus between IBM and the vendors. Although the vendors could use it to extract to, they usually extract directly to their own networks.

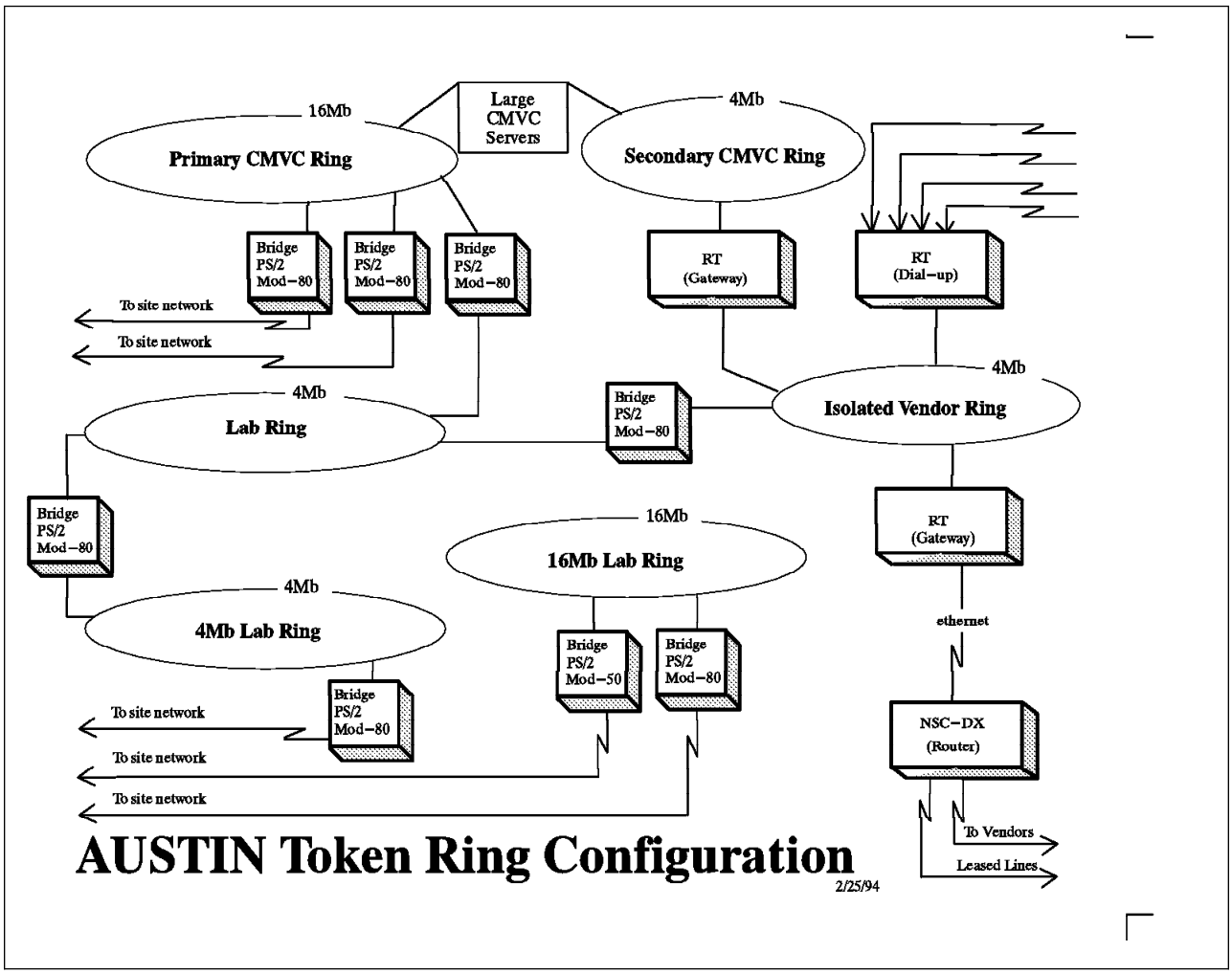


Figure 11. LAN Configuration at IBM Austin

5.2.3.1 Remote Resources

Users are literally spread around the world and access the server from other CMVC clients located in countries such as Israel, Great Britain, Japan, France, Korea, Australia, Egypt, Canada, Italy, Switzerland, and Germany. The overall performance of access to the CMVC server across continent boundaries is directly related to the performance of the underlying network. The TCP/IP topology allows a response time of about 400 msec when the `ping` command is sent from Europe to IBM Austin.

Many other development groups are located in the United States and access the servers through the IBM internal network from cities such as San Jose, Kingston, Boca Raton, Rochester, Dallas, Roanoke, Endicott, Gaithersburg, Hawthorne, Austin, Palo Alto, Raleigh, Kingston, and Tucson. Other users access the CMVC servers through dial-in lines such as software vendors and non-IBM organizations such as OSF.

IBM sites or vendors that have an adequate network access use local CMVC clients to connect to the CMVC servers in Austin. Sites with slow connections (for example, when using slow dial-in lines or wide area networks between continents)

use the command line interface of CMVC to get the data to and from their working space on their local machines.

5.2.3.2 Backup Concept

To ensure the maximum degree of availability a thorough concept for backup and archiving is set in place. Incremental backups of the CMVC data are saved daily, and full backups are taken during low system load time during the weekends. The time required for a daily backup is less than 40 minutes for the biggest server and family, and a full backup takes up to four hours of elapsed time. As the servers are running CMVC and nothing else, the backup only addresses the CMVC data and the associated SCCS files. No system backup is taken.

5.2.3.3 Dedicated Servers

Seven CMVC servers are used online in the production environment. In order for these servers to run pure CMVC, any other task related to support CMVC is put on other machines. Any customization done to the CMVC installations is tried and tested on test machines that run a clone of the production environment. If the test passes, the internet addresses of the original and the clone machines are toggled, and the new machine running the new customized version becomes the production server. A similar concept is also used for a regular database reorganization. There is no way to reorganize the database in a dedicated mode, as the servers need to be up and running around the clock. Periodically the entire contents of the database are extracted and reloaded onto a clone system. Once the clone is up again, the server addresses and clone addresses are toggled, and the roles change. IBM Austin experiences a performance improvement between 10% and 15% each time the database is reorganized.

Another dedicated machine is used to gather data from CMVC at night to be used for management reports and similar charts. This server would run shell scripts to get CMVC reports and process the data further on to produce the proper listings, charts, or reports.

5.3 Centralized, Formalized Support Structures and Tools and Specialized End-User Roles

An organization such as IBM Austin has put in place a very special organization to support the various requirements of the CMVC end users. This was done to reflect the very specific skills needed and the amount of support that has to be provided to the users.

5.3.1 Help Desk for End Users

The total number of users on the different servers and families at the IBM Austin site is certainly impressive and requires well-organized support. CMVC is used almost round the clock from worldwide locations. All types of questions are bound to arise as for every heavily used application.

IBM Austin has put some effort into providing a help desk to assist users in their day-to-day operations. The help desk deals with all problems the user may encounter with the system, from problems directly related to CMVC screens or processes, to access permission, to network-related issues.

Developers at IBM Austin wrote an online application with a GUI to collect the questions asked by phone and classify them into different areas. If the answer is

known, the user gets immediate help. If not, the question is passed to the area specialist for resolution. In addition, the user has access to a sort of information database containing the most frequently asked questions together with the solution to the most common problems. This database is used for reference and maintained by the help desk staff. With this background, about half of all questions can be answered directly by the help desk staff, leaving the support specialist time to solve more complex or new problems.

5.3.2 Tool Developers

Because of the extensive usage of CMVC, a separate support group has been established to deal with the CMVC extensions. The activities are requested by the development groups and discussed in periodic meetings between the two organizations. This development support group makes the following available for end users:

- User exits to increase the CMVC functionality for the specific IBM Austin code development process
- Canned queries for getting management data out of the CMVC database
- Shell script to be executed as batch jobs for repetitive tasks instead of going through the CMVC GUI
- Utilities to extract data into the management database, which is a collection of CMVC raw data to be used for more sophisticated reports
- Graphical front end to select the data from the management database and organize it into tables or graphs.

Some of these additions were first built on top of Orbit and have become part of CMVC. See 5.5, “Extensions to CMVC” on page 90 for a description of the back-end and front-end extensions that have been written to interface with CMVC.

5.3.3 System Administrator

Because of the network topology at IBM Austin, the tasks of the system administrator are very demanding.

The servers are purely CMVC servers, so the backup strategy is centered on the CMVC data only, with no need for full system backup. To optimize backup times, they do an incremental backup every night, which takes about 40 minutes, and a 4-hour full backup during the weekends. A full automatic backup strategy making use of ADSM is under study.

Because system availability should be very high, they monitor the system log to prevent any hardware failures that could impact development. Because the topology is dymanic, it installs new clients whenever necessary, allowing the network to grow further.

The performance of the CMVC system is highly critical, so they periodically reorganizes the database, which improves performance by about 10% to 20%. For the same reason they try to optimize each batch job according to the data available from the CMVC audit log, which keeps track of each transaction.

One of the key tasks in 1994 was the migration of CMVC Version 1 families to CMVC Version 2. The system administrator along with the family administrator worked on migrating eight families to the current CMVC Version 2.2. Currently (December 1994) more than five families are to be migrated the first quarter of 1995. The transition must be of minimal impact to development due to advance

planning and communication via notes to the developers. All migrated families were created with default settings with the option of adding user exits and some optional new enhancements on an as-needed basis.

5.3.4 CMVC Family Administrator

The task of a CMVC family administrator begins with the user support. Because of the extremely high number of users, support tools have been implemented to help assign userids and administer them. These online tools are directly invoked by the user. The typical tasks of a family administrator are:

- Project rollout

  Set up the component structure together with the project team, user authorization, and user education

- Ongoing user support

  Answer the usability questions that may arise, keeping up with new CMVC versions

- Customization

  set up the choice list, notification lists, host lists introducing the CMVC extensions, back-end and front-end tools into a specific project environment

- Technical support

  Provide documentation for accessing CMVC client code and guidelines on technical issues related to the CMVC configuration.

5.3.5 Education

The end user education on CMVC-specific issues is extremely important for a smoothly running development environment. Many materials are available for a new development team member who wants to become familiar with CMVC.

The education material is available in the form of presentations or a short introduction to the CMVC concepts. Additional training material and classes have been prepared and targeted to the specific kind of interaction the user is supposed to have with CMVC. This is defined by the specialized user roles in use at IBM Austin.

5.3.6 Specialized End-User Roles

Specialized roles include coustomer support, team leader, development manager, project manager, planner, build and release manager, software developer, tester, and information developer.

Each of these roles is accomplished not just by one or two individuals but by large groups of people. This aspect points out how CMVC helps to satisfy the sophisticated requirement for effective project communication through its automatic notification mechanism. The process implemented at IBM Austin under CMVC V1 is equivalent to the configurable process offered in CMVC V2. The subprocess-related activities using approval, fix, and test records foster additional communication among team members.

Each user role presents a specific form of interaction with CMVC, and not everyone is bound to have deep knowledge of how all of CMVC works to complete a specific task.

5.3.6.1 Customer Support

The customer support group is responsible for getting the defects from the customer installations and forwarding them for resolution to the development group.

The customer support group uses a system called RETAIN to input defects, but it wrote a CMVC utility that transfers this information and opens a CMVC defect on the affected components and tracks for all releases in use.

Development must ensure that the defects discovered in one release are corrected for all releases actually supported at the customer sites as well as for all releases under development. It reviews the tracks opened by the CMVC utility and takes the appropriate action for all supported and future releases

Customer support does not need to know anything about CMVC because everything is done automatically.

5.3.6.2 Team Leader

The team leader has to master every aspect of CMVC because he or she will go through all CMVC steps, from configuring the CMVC component hierarchy for a new project, to setting up userids and their access authority, to defining releases and levels, to following the process of a defect or feature from open to close and monitoring the status of the tracks.

5.3.6.3 Development Manager, Project Manager, and Planner

The development manager, project manager, and planner use the information in the CMVC database primarily to infer the status of the project and make projections for future development.

What these groups need to know is the granularity of data and the range and semantic of the single fields they can get out of CMVC. Thye will use this data as input for their status reports, quality projections, and resource estimations.

See 5.6, “Project Management” on page 96 for details about how project managers use CMVC.

5.3.6.4 Build and Release Manager

Build managers should have a detailed knowledge of the status of the defects and the organization of the files containing the code into releases, levels, and tracks. They can extract the right pieces of code from CMVC to build a module to be passed to the tester. They also open defects against the component that caused build to fail.

5.3.6.5 Software Developer

The software developer uses CMVC primarily as a version control and problem tracking system for the group of files on which he or she is working.

A deep knowledge of CMVC concepts (processes, component hierarchy, releases, levels and tracks, problem status and overall project organization) is required to perform this job. Additionally detailed knowledge of CMVC customization, extensions, and front- and back-end tools is required to take advantage of the whole development environment.

5.3.6.6 Tester

The tester basically interacts with the problem tracking function of CMVC. During the test defects are discovered and the fixes proposed by the software developer are verified. Testers create defects against the code as well as against product documentation.

Testers are familiar with CMVC concepts like releases, levels, tracks, processes, and problem status. The tester's task is highly dependent on interproject communication with the developers, for example, for clarification of the results of the test compared with the specification of a certain command or function.

5.3.6.7 Information Developer

The information design and development department uses CMVC:

- As a repository of InfoExplorer library images just prior to shipment as part of the AIX code installation image.
- As a mechanism for recording and tracking problems in the online and hardcopy documentation for AIX.

The data is organized into the same family as the AIX code because:

- The defects opened against a document may need to be reassigned to a code component upon evaluation, or vice versa.
- A defect in code may be discovered by information development people and code developers may discover a defect in the documentation.
- Some defects may affect both documentation and code and need to reference each other.

The information development process consists of the following steps:

1. Documentation is written using Interleaf on a RISC System/6000. The Interleaf native format is stored in Interleaf version control. At present there are about 20,000 files. CMVC is not used for documentation version control because this function is provided within Interleaf in an integrated and easy-to-use way.
2. The documents are converted into PostScript files and sent to the printer. There are about 40,000 pages in 150 volumes.
3. The documents are then translated into InfoExplorer files by InfoCrafter. InfoExplorer images are stored in CMVC components that represent the Info Libraries (.rom and .key).

Defects for these components are only for errors in images, not for the content of the documentation itself. Each document defect is entered as hypertext link on a certain defect number. All components are at the same level, as supporting skeleton for problem tracking only.

The documentation components are stored in a hierarchy with one component for each book title. Sometimes you may have multiple volumes for a title. There are about 150 components and there are no files associated with these components. Defects are opened against common document sources for both hard and softcopy. There are different ways to ensure that defects related to document and code get linked:

- Create two defects, each referencing the other through a reference field . If one defect is closed and the other is still open, there is no automatic way to carry forward the reference.

- Create only one defect, but a track in each release (that is, one for documentation and one for code release). The limitation is that you cannot close the defect without the other track being committed.
- Add additional configurable fields for cross-reference to enable the recording of related defects.

Defects can be opened by everyone and are reassigned from the top level to the correct component as needed. The defects against documentation are processed as follows:

1. A defect is accepted or rejected on the authority of document owner.
2. The defect is reviewed by the "must fix" board unless it is challenged
3. A priority similar to code is used.
4. The information developer gets UNIX mail whenever a defect is opened against a component owned by him or her
5. Developers move the defect to Fix status only when an interim release of the InfoExplorer library is built
6. No version control information is lifted from Interleaf to be recorded into the defect
7. The defect opener verifies the defect based on its appearance in the interim InfoExplorer library, doing a search by defect number.
8. About 5% of total defects are opened against documentation components.

Information developers use either the GUI or VM client for open, accept, fix, and reject defects. They use shells for user-specific queries.

For documentation they proceed as follows:

- No PTFs are generated for InfoExplorer; the complete new images are shipped.
- Books are occasionally reissued between releases with a new suffix number (for example, XXXX-XXXX-01, 02).
- A complete book is delivered, no substitution of single pages
- The schedule for publication is not identical to the schedule for code, that is, the books do not have not to be ready as early as the code.

5.4 Customizing CMVC

The sophisticated use of CMVC at IBM Austin becomes obvious when one looks at the different points where CMVC allows customization. This section shows the various customizations applied to CMVC in order to fine-tune CMVC to the development process that is being used.

5.4.1 Configurable Fields

IBM Austin uses several CMVC families, some of which are still running with CMVC version 1.1, others already at version 2.2. The plan is to migrate to the new version within the next few months. The problem with the back-level version of CMVC is that several features of CMVC, such as the possibility of adding additional configurable fields to the CMVC database, are very important for a sophisticated use but are not available.

Therefore, if CMVC cannot be upgraded to version 2.2 because the schedule does not allow it, the lack of configurable fields must be solved by overloading existing fields with a specific syntax and an associated semantic (see 5.4.2, “Overloading of Fields” on page 87).

This is not a nice implementation, but it was the only way IBM Austin could still stay at version 1.1, but already tune CMVC to its process requirements. If the corresponding family is moved to CMVC version 2.2, a configurable field could be used to indicate the same information.

Other planned fields are:

- A field to store the lines-of-code count for source files in the `File` table
- A field in the `defect` table to indicate that the defect also affects the publication

5.4.2 Overloading of Fields

Some of the families at the IBM Austin site are still using CMVC version 1.1 and cannot use configurable fields. Instead, these projects have defined a convention to use the predefined fields in a specific way to associate certain semantics with it. The allowed values of the different fields thus become highly customized to the process of the projects that are using the fields.

5.4.2.1 A Field Triggers Other Automated Procedures

The *note* field of a defect is used at the IBM Austin site in a specific and unusual way. It is used to indicate that the defect does not apply to a particular release of the code under development. This information is used by a batch routine, which would create tracks for reach release for each defect unless the note field indicates that the defect does not apply for a particular release. This batch routine is run at night or during times of low system load and inspects all defects.

The information is added to the note field after the developer answering the defect has checked that a code change is not required for fixing the defect for the new release. Thus when the batch routine runs the next time, it would not create a track for the release again, because the note field indicates that development has already determined that the code for this release does not require any modification in order to fix the defect.

5.4.2.2 One CMVC Field Stores Multiple Logical Fields

The note field of a defect is also used to decide whether or not a defect should be deferred or implemented in a particular driver. The developer recommends what should happen with the defect, but a so-called *MustFix board* actually decides whether or not a defect is going to be deferred. To prepare for the board's decision, specific information is put into the note field according to a template. This information is provided by the tester and the developer. The logical fields that are put into the note field are:

- Contact person and phone number
- Symptom the customer or tester experiences
- Impact to the customer or tester
- Impact to the entire system
- Pervasiveness of the problem
- Injecting defect or feature if known, and its release
- Known workarounds
- Proposed solution and alternatives
- Size and risk of implementing the fix
- Build complexity (amount of files that need to be changed and files that need to be rebuilt)
- Code reviewer.

Once these fields are filled in, the MustFix board decides on each defect during a periodic board meeting.

5.4.2.3 Allowed Values for a Field Differ over Time

The use of the *priority* field of a defect is also a good example of how the contents of a CMVC field is used throughout the development process over time for project management purposes.

In the development process used at IBM Austin two specific checkpoints (MustFix and StopShip) affect the use of the *priority* field.

When a project reaches the MustFix stage, the decision about whether or not a particular defect is to be deferred will be taken during formalized reviews from the *MustFix board*. This review board basically decides whether defects become *Deferred* or *Approved*, once they are accepted by development to be valid defects. The MustFix board uses the information in the *note* field of the defect (see 5.4.2.2, “One CMVC Field Stores Multiple Logical Fields” on page 87) to decide which defect is deferred and which is not.

Some time later in the development process, the project leaves the MustFix stage and enters StopShip stage. At this point, the deadline of the project is close, and only minor changes to the code are allowed unless the problem is of great importance and must be fixed under all circumstances even if this implies resetting the schedule. All defects that show Approved in the priority field are inspected again. The approved defects are basically divided into three categories:

- Deferred defects, which are moved to the next release
- StopShip defects, which must be implemented in the current release even if this implies a change of the schedule
- DevServed defects, which are implemented and shipped to customers as PTFs rather than becoming part of the official product tape.

Figure 12 on page 89 shows the evolution of the contents of the *priority* field over time and the modifications made at the various project checkpoints.

The closer the project comes to the schedule deadline, the more important it becomes for management to be able to estimate the remaining effort precisely. CMVC is one of the tools to provide that information. The reports used for the review board decisions (see 5.5.2, “Complex Reports” on page 93) are run daily to ensure that management decisions are based on up-to-date data.

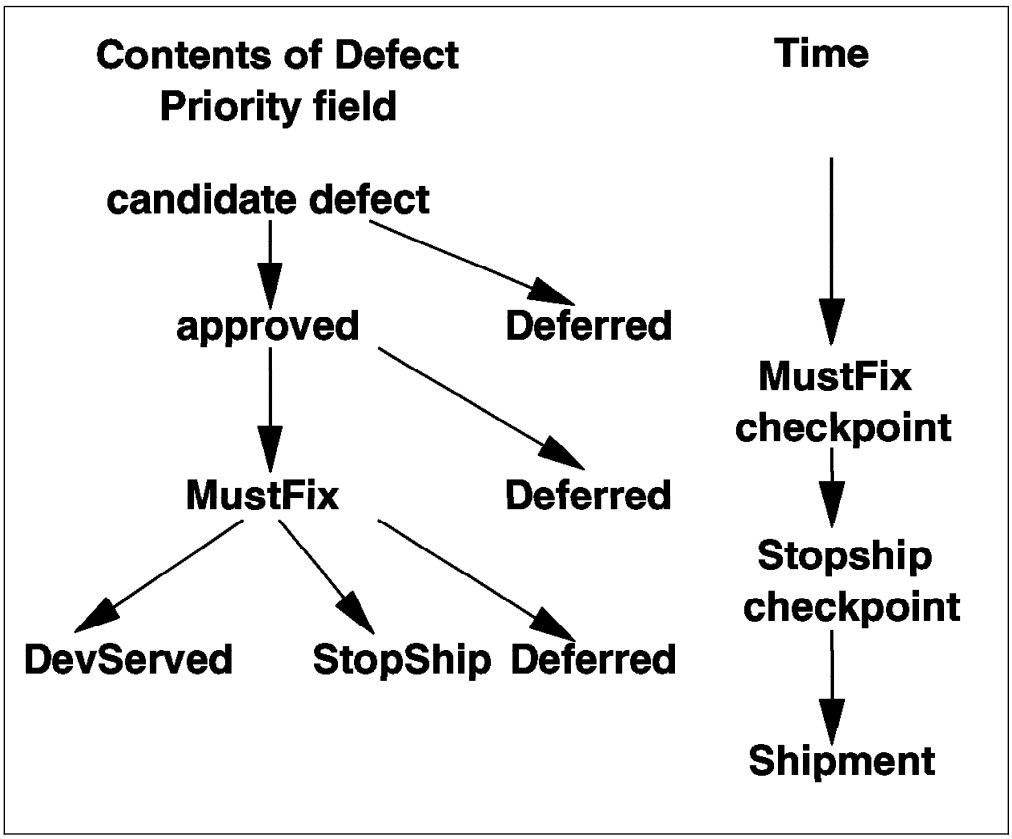


Figure 12. Defect Process Flow Based on Priority Field

5.4.3 Choices

The different families in active use at IBM Austin have different configuration tables associated with them. This maps to different lists of choices for some fields on the GUI. For example, the list of choices when accepting a defect consists of 10 different alternatives when CMVC is installed. One family set up at IBM Austin redefines this list of choices to match the specific needs of the development process and the needs of the project and has defined a total of 16 alternatives. A second family actually reduced the default list and offers only six alternatives.

Each development group thus uses choices that match the different project requirements, the different terminologies, and the specific processes it uses.

5.4.4 User Exits

The user exits listed below are in use at the IBM Austin site. Some of the families are still using CMVC version 1.1, so the ways of taking advantage of the user exit capabilities of CMVC are limited.

- When a user does a Verify -reject or a Test -reject, he or she triggers a user exit that automatically opens a new defect and copies the old data and text from the original defect. These scripts automatically open a new defect for the user using the information contained in the one that tested bad. These two routines are about 85 lines of shell code.

The advantage of this user exit is that all information from the existing defect is automatically copied to the new one; the user does not have to manually retype the information.

- When a user issues a File -create and File -checkin command, he or she triggers another user exit. This user exit also is a small shell script of approximately 40 lines of code. It calls another program to verify that the source file to be checked in or to be created contains valid prologs. The purpose of this user exit is to enforce some development standards. The user exit does not prevent the file from being created or checked in if the prolog is invalid. However, if the prolog is invalid, the user exit automatically opens a defect for this noncompliance.
- Another user exit is currently being developed. This exit will be invoked whenever a file is created or checked in, and it will count the number of source lines in the source file. The line count will be stored in the CMVC versions record, which has fields reserved for this information.

  The line of code count can be used to gather productivity statistics such as error rates per line of code and fixing rates. This kind of information is useful for follow-on projects, which can base their assumptions for productivity, maintenance effort for fixing and test, and schedule estimations on real-life data from previous releases or projects

5.5 Extensions to CMVC

IBM Austin has developed quite a few extensions to CMVC that interact with CMVC as front-end or back-end tools. These interface programs are used to further process the data that is stored and maintained by CMVC, and this information is used in appropriate places in the IBM Austin development process.

5.5.1 Back-end Tools

IBM Austin uses various back-end tools and procedures to interface with CMVC. These tools are used for different purposes and operate on different families, some of which run with CMVC version 1.1.

5.5.1.1 Transaction Analysis

CMVC logs all activities and transactions in an audit log for each family. This log contains information about each successful and each unsuccessful transaction and therefore provides a history of the CMVC family. For each transaction information such as the component or file name is recorded against which the transaction was performed.

IBM Austin has implemented analysis programs that are run periodically every day, week, and month. These programs group the history data from the audit log and generate report tables as well as graphical representations of the data. See Table 3 on page 78 for an example of such a transaction summary. See Figure 13 on page 91 shows the distribution of CMVC transactions over the different hours of day.

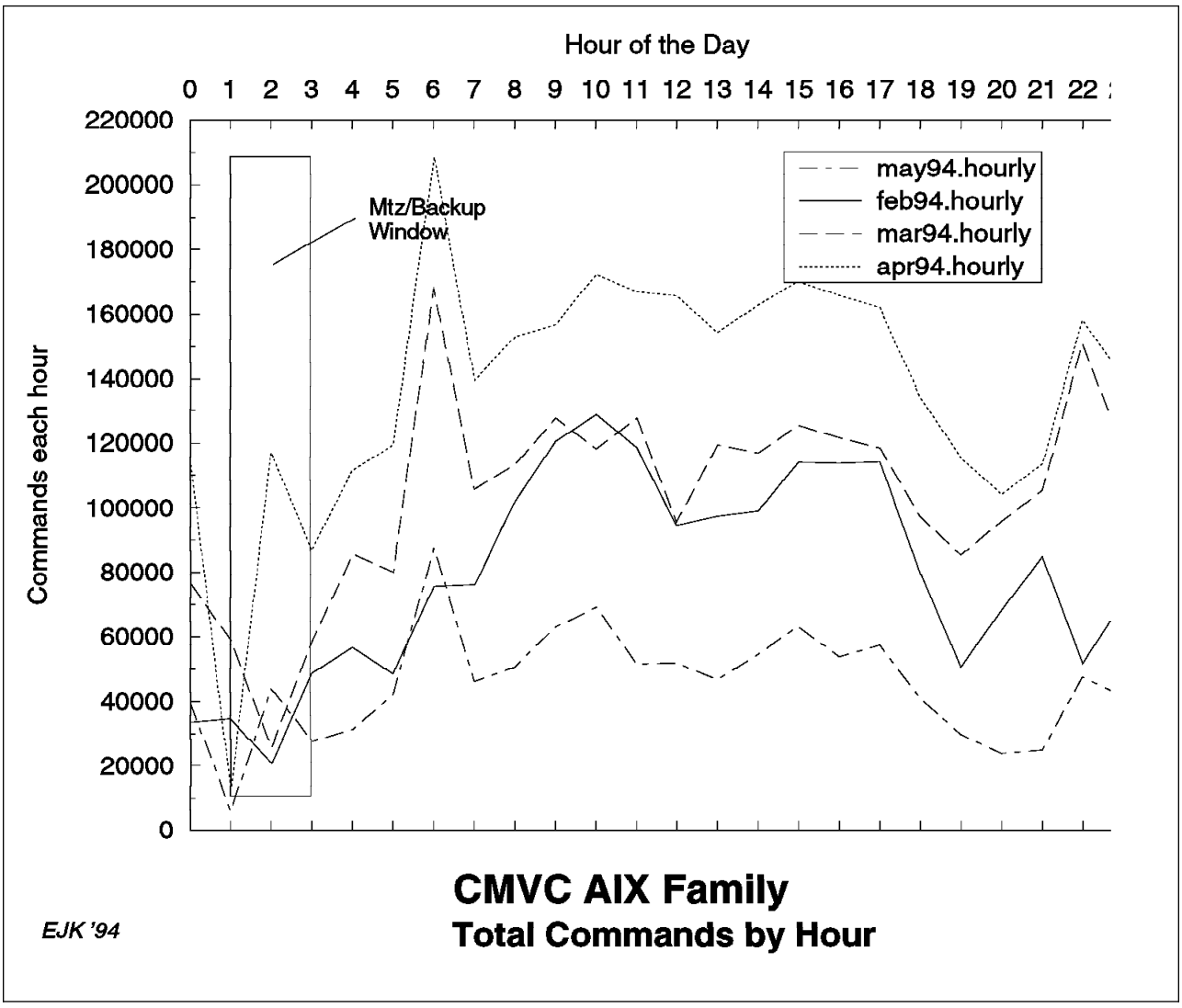


Figure 13. Distribution of CMVC Transactions over the Day at Austin

The transaction summary is used for at least two purposes:

- Those commands and reports that are issued most are selected for further investigation. Duplicate reports are identified and run from a central support organization rather than from each group individually, so system load can be reduced. Also, the central support group takes over the charge of maintaining and enhancing these central report tools. This is an important point, as nearly 60% of all CMVC transactions are calls to the `Report` command.
- The transaction summary data is also used to tune system load over time. Graphical representations of the transaction history show the number of transactions during the day, and thus periods of low CMVC usage are identified. Time consuming CMVC batch jobs such as build procedures or lengthy report generating procedures are started to run at times of low system load to minimize the impact on interactive CMVC users during the day.

5.5.1.2 Automatic Limit Checking

A key requirement of the CMVC operation at IBM Austin is an availability of 100%. It is therefore of utmost importance to recognize potential system administration problems such as shortage of external disk space and shortage of file systems long before those problems cause the failure of CMVC commands. The problems must be detected before they even occur.

To detect problems early, the system administration group has set up a couple of automatic batch jobs as crontab entries to perform early warning functions for the following critical situations:

- Check for shortage of disk space
- Check for shortage of disk storage
- Check whether the CMVC environment including all database demons can be started correctly again after a backup job

A planned enhancement for this early warning system is to automatically notify the beeper of the system administrator by sending a paging message. Right now, these automated jobs send e-mail to the system administration staff.

5.5.1.3 Track Creation

The problem that IBM Austin faces results from the fact that previous releases of AIX, such as AIX 3.2.5, are in the maintenance phase whereas new versions or releases are concurrently under development.

When a defect is reported against releases under development, the same defect may also apply to the same source code files in later releases that are still under development. CMVC uses tracks to follow up on the resolution of defects in multiple releases at the same time, so the development group needs to make sure that tracks are created for each release where the same defect may need to be fixed.

This is done by a nightly batch job, which is triggered through a crontab entry. The batch job looks at all of the defects and creates tracks for each release where the defect may occur. The source files for the multiple releases may no longer be common, so the developers need to look at all code levels that are in maintenance or development to decide whether the defect is also valid for the particular release.

If the defect does not apply to that release (for instance, because the new code for a release under development may have caused a rewrite of the code), the corresponding track has to be closed. In addition, the same batch job must not recreate the track for that release again the next night.

To do this, a note added to the comment field of the defect indicates that the defect does not affect a certain release. The nightly batch job looks at the comment field to decide whether or not to create tracks for the various releases.

5.5.1.4 Automated Code Review

The AIX development groups perform code inspections as part of their regular development process and use a tool to support in this effort during the test phases.

The tool is used when a defect has been reported by testers and the responsible developer has accepted the defect. The corresponding tracks are created and end up in fix state, until the developer is done with the code modifications.

Once code modification has been completed, a code inspection cycle takes place. The code is checked into CMVC while the track is still in fix state. The developer contacts the reviewer by phone, e-mail, or by knocking on heaven's door. The reviewer to perform the code inspection then invokes the tool along with the defect number. The tool then gets the original source file and the one including the modifications from CMVC and performs a `diff` command. An editor then shows the code differences, so that the reviewer is presented with the changes that the developer has made.

Figure 14 shows the syntax for the invocation of the code review tool.

```
Usage: review [-ifl] [-p diffprog ] [-r release] [-w dir ] defect/feature
       review [ -h | -? ]

   flags:
           -i: interactive mode; review each file only if requested
           -f: review a CMVC feature rather than a defect
           -l: show long version of CMVC defect or feature
           -r: only show tracks for specified release
           -w: working directory, default is /tmp
           -p: program to use for diff rather than "sdiff -w 160 -l"
               (if args to diff program, put in quotes as in "diff -e")
           -h: see -?
           -?: see -h
```

Figure 14. Command Syntax for Tool Used for Automated Code Review

The tool shows the changes made by the defect or feature based on the tracks dropped in CMVC and asks for approval of these changes from the reviewer. Upon approval, an approval note is added to the CMVC defect or feature and mail is sent to developer.

This process could be further automated by automatically creating a tool to send a request for review after the developer has checked in the source file again. Some people have already set up their own mailer scripts for this. The developer moves the track to integrate state only after approval. Once the code is in integrate, it can be picked up for a build.

By introducing this automated code review, the development group estimates that about 10% to 20% of bad code fixes are found and avoided before the code actually goes to the test groups. This number strongly underlines the importance and preference of defect prevention rather than defect testing.

5.2.2 Complex Reports

The transaction analysis reports show that the CMVC `Report` command is by far the most important command (see Table 3 on page 78). In fact, the data generated from the CMVC database is the most important instrument for gaining management insight into the status of the projects and getting information about trends, the effort for the current work load, and the potential effort in the near future.

A very sophisticated and automated processing of these reports is therefore put in place, and the procedure for getting the data from CMVC, generating reports, and creating graphical charts of the reports is basically fully automated. The huge size of the underlying database (the largest family controls more than 600,000 files) results in a lengthy run time of these reporting procedures, so these are schedules to run at times of lowest system load.

The most complex CMVC family is used for development and maintenance of the AIX operating system. First, programs gather CMVC data using the CMVC `Report` command, then postprocessing programs work with the extracted data and load the data into a management database (MDB). Some other small tables in that MDB describe dependencies that are not contained in CMVC such as the IBM Austin organization setup of areas, groups, and departments as well as information about which developer is working in which department. Several reports as explained below focus on departmental or similar grouping criteria and that data is not covered by CMVC.

Several utilities already have been written against this database, and these programs are used for quality projections and further defect prevention analysis (5.6, “Project Management” on page 96). The utilities are mostly written in perl and other native AIX or UNIX programming languages.

The following reports are generated automatically:

- Deferred defects reports, which includes severities of the defects. This report is grouped by department.
- Summary reports, which show defect statistics about arrivals, closes, and changes since the last week. These reports also show the total number of defects in the various states grouped by severity or development area.
- Master reports,which show more details about defects such as component, severity, priority, release, owner area and the defect abstract. These reports are generated for defects that:
  - Must be fixed
  - Are potential defects with severity 1 or 2
  - Are potential defects with severity 3 or 4.

  The purpose of the list of potential defects is to gain management insight into the potential work load of the development groups.
- Blocking reports, which show those defects that block (prevent) parts of the test to be continued. The reports identify the test area that cannot continue, such as performance test and usability test for each blocking defect. An indicator of the type of blocking of a defect is also added to all other reports that show a list of defects.
- Master reports, which are grouped by groups, departments, and areas.
- Arrival and closure reports, which summarize the difference of accepted defects that must be fixed between the day of the report and the previous day. The same report is gathered for potential defects. From these reports, the corresponding arrival rates are calculated and ordered by area.
- Test/verify/returned reports, which are used by the test groups rather than the development groups. These reports describe those defects that are in *test* or *verify* state, as well as those that are rejected because the fix did not solve the original defect to the tester's satisfaction. A user exit automatically creates a new defect with the same contents for the abstract, severity, and the like once a fix for a defect is rejected.

The reports discussed above are generated from central staff personnel and mailed in softcopy or hardcopy format to the various groups that are interested in them. In addition, graphical representations of some of the reports are generated using

various tools, for example, to show the number of defects along with either their severity or status over time (similar to those used at IBM Tucson as shown in Figure 4 on page 61).

In addition, a set of front-end utilities is available to generate reports and analysis graphs. These utilities are used for instance by the individual developer, who may not be interested in the defects for the entire family or for certain areas, but just in those where he or is the owner of the components that are supposed to be the cause of the defects. Similar utilities exist for breaking down the master reports into the different departments or components.

A second area at IBM Austin works with product development of OS/2 software. For the OS/2 projects at IBM Austin, the following reports are generated using OS/2 REXX command files that issue commands against the command line interface of the OS/2 client:

FMTRPT a batch procedure that provides a formatted report of defects for given components. The report includes the values for prefix, number, component, state, origin ID, owner ID, severity, age, phase found, answer, and abstract.

LSTFILES returns a list of files in a given component

EXTFILES extracts all files for a given release and component

CNTPHASE counts the number of defects per phase found choice. Defects in canceled state (except those with an answer code indicating future resolution or limitation) are not included.

CNTVAL counts valid defects for given components

CNTINVAL counts the number of invalid defects for given components

CNTSEV12 counts the defects with a severity level of 1 or 2

PRTTIME provides an average count of the number of days it took for closed defects to go from open to last reassign for defects with a component equal to one of a given list of components.

TURNTIME provides an average count of the number of days that it took defects to go from OPEN to CLOSE, if the PHASEFOUND is equal to fvt and a component equal to one of a given list of components.

CNTACTIV provides a count of the active defects (not including canceled and closed defects) for given components

CNTOPCL7 counts those defects that were opened, closed, or canceled in the last seven days

CNTANSW counts the defects per answer code that are not in OPEN state for given components

CNTSEVST counts defects per severity code and state code for given components

CNTSTATE counts defects per phase found and per state for given component

5.5.3 Front-End Tools

CMVC offers a GUI as well as a command line interface, which can be used to develop other tools acting as front-end tools to CMVC. Among others, IBM Austin developed the following two front-end tools to interface with CMVC:

CMVCCMD This is a batch procedure that takes a list of CMVC commands from an ASCII flat file and executes them using the CMVC command line interface. This tool is used for the OS/2 projects, and it is implemented as an OS/2 REXX program. Errors that are encountered from CMVC are written to an error log file and may be checked later by the user.

The purpose of this tool is to gather repetitive CMVC commands together rather than having to type them again and again. Also, the tool allows CMVC commands to be submitted as batch jobs in parallel to other OS/2 tasks.

RETAIN Interface This is a system very similar to ZAPAR at IBM Tucson (see 4.7.2.1, “ZAPAR” on page 63). It is used to create and process CMVC defects against code for IBM program products that are out in the field. The service organizations and field personnel report these defects against another database application (RETAIN), and the developers use the RETAIN-to-CMVC interface tool to feed the CMVC database with information from RETAIN and vice versa.

Whereas the ZAPAR tool runs on VM, the RETAIN interface tool IBM Austin uses runs on AIX. It uses an IBM internal emulator to interface with RETAIN, which is running on VM, and uses the command line interface of the CMVC clients on AIX.

5.6 Project Management

Project management is another heavy task at the IBM Austin location. As with other functions in the IBM Austin environment, collecting data for project management has evolved during the last six years with respect to both the granularity of the data extracted out of the RDBMS and its graphical representation.

An important aspect of software reliability is acquiring actual and valid data. In this context, the data is the number of failures of the software product under test or usage as they occur over time. This definition is met at the IBM Austin site because it used CMVC from the very beginning of its operating system development.

The data extracted is organized in reports with different detail refinements to be used by team leaders on a single project group, by management on several development groups, and by middle management on a group of departments. User programs are available to break down the data by CMVC ID and by component. The data representation is provided under different formats, such as tables and two-dimensional graphs.

Because the development of an operating system is a very sensitive, high impact activity, these reports are generated on a nightly basis and are immediately available for review in the morning. Formal review meetings have been established, and the very same reports are used for monthly measurement meetings, which analyze the quality of the ongoing development, or for the program

assessment review, which is called during the last month of the development of a certain release. Based on a subjective judgment of the quality management must decide whether the product is ready to be shipped.

5.6.1 Monitor Project Status

Just to take into consideration the AIX operation system only, CMVC consists of one family containing the code of at least 5 releases and includes all the the components which are directly developed in IBM Austin and all the parts which are been worked on by different vendors in locations spreaded all over the world. All the parties use this CMVC family as common configuration management system and even more important from this point of view as their problem tracking system. Doing so they build a common database and share a common understanding of this software life cycle. From the data stored into CMVC they can get all the information to monitor the progress of the project with the accuracy and reliability they need.

5.6.2 Ensure Project Adherence to Schedule

Some reports have been designed to get a measurement of how close the project is to the reference project schedule. By collecting data on failures and by classifying them in different types, the project manager can estimate, with a reasonable degree of accuracy, the number of defects, or possible failures, remaining in the software product. Having access to this kind of information helps the decision maker decide whether a product is ready for release to the market. A good example is the arrival and closure defects report, which shows the difference between today and yesterday of the number of open and closed defects. Another significant example is the so-called breaking defects report, which shows separately the defects, belonging to the categories of build breaks defects, blocking test defects, and blocking performance test defects.

5.6.3 Calculate Software Quality Metrics

It is possible to get data that represents most of the useful parameters for quality metrics out of the huge CMVC database. This data refers to the releases that are under current development as well as to those installed at customer sites. The data can be used to feed mathematical models and correlated under different criteria to get information with a better degree of reliability. For instance, IBM Austin uses a parato analysis, which states that 80% of the defects are to be found in 20% of the code. This helps identify the most critical components.

5.6.4 Measure Productivity

Because the data can be broken down per department, it is possible to get a qualitative as well as quantitative measurement of productivity. This leads to a better knowledge of the development results and it is used for quality projections as well as for quality measurement on current development.

5.6.5 Adjust Resource Allocation and Planning

Because the data is kept on different releases it is possible to correlate information. Based on the problem reported by internal test groups on the development version and by the customers on the versions currently in the field, they can identify the critical components in the product. This identification is used to reshuffle resources on the current development version to meet the target shipment date and to get more accurate planning for the next version. Furthermore, the CMVC data is

correlated with independently gathered data, such as customer satisfaction ratings for quality projection for upcoming releases. Based on the complexity of the code and on its failure rate, it is possible to predict which components are likely to absorb the most resources if major changes are expected in a certain area.

5.7 ISO 9000

IBM Austin uses CMVC to achieve ISO 9000 compliance. A particular CMVC family (the so-called Admin family) is used exclusively for ISO 9000 purposes. It has more than 3000 userids and is still being controlled by a CMVC server running CMVC version 1.1. Some departments store their department operating manuals (DOMs) in the Admin family, but this is not yet done consistently because not everybody in the IBM Austin lab has access to the CMVC server machine that maintains the ADMIN family, so not everybody would have access to the DOM for his or her department. This, however, is one of the requirements of the ISO process, so some of the DOMs are stored online on VM.

Problems during the development process that are caused by faults of the process itself are reported as defects in the ADMIN family. This allows one to differentiate between a problem in the code and a problem caused by a fault in the process. A sampling of defects against the development family is used for defect prevention process (DPP) analysis. Developers look for the causes of the defects and put information about the cause into the Note field of the defects (another example of the overloading of the Note field as explained in 5.4.2.2, “One CMVC Field Stores Multiple Logical Fields” on page 87). IBM Austin plans to introduce a new configurable field, the DDP done (Y/N) field, and use this information to change the process, control vendors, or manage developers. It also wants to cross-reference to a defect in the Admin family from the cause, if the cause really addresses the process.

Chapter 6. Conclusion

Once a thoughtful look is taken at application development in a team programming environment, the advantages of investing in a solid, generalized, and flexible software configuration management tool is indisputable. The question is not whether you need it, but which tool best supports your needs?

Experienced software developers can make intelligent guesses about the usefulness of a sophisticated software configuration management tool by studying sales brochures, examining product manuals, and observing demonstrations. They can also evaluate a tool's applicability to their unique situation by investing the time in an exploratory exercise of the tool in their environment. An intermediate step, however, is the examination of the experiences of other organizations who have already used the tool to accomplish real work. As this book shows, CMVC is perceived by its users as an industrial strength, commercial CM solution that meets and exceeds their expectations.

It is significant that the company that produced CMVC depends so absolutely on CMVC in its many development laboratories. IBM has widely standardized on the use of CMVC for application, scientific and engineering, and systems software development targeted to a wide variety of computer platforms. This standardization on CMVC has emerged within separate development organizations, in most IBM divisions around the world, because each group independently evaluated the benefits and advantages of CMVC.

CMVC customers apply CMVC to solve problems directly related to software configuration management but have also discovered its usefulness to control and manage documents, drawings, and many other deliverables. They are using CMVC to control business application development, engineering and manufacturing software, operating system, and software products. They use CMVC to manage source code written in assembler, traditional 3GLs, 4GLs, and object-oriented languages.

Customers find that CMVC helps to generate and enforce standardization at many levels within a software development organization. Many customers use CMVC to support their attainment and retention of ISO certification. Likewise, CMVC can be used to help a company conform to computer program development and management requirements such as those promulgated by industry bodies such as the Software Engineering Institute (SEI) or by government agencies, such as the U.S. Department of Defense or NATO.

CMVC users customize CMVC to meet their unique needs using the built-in features it provides and extend CMVC with tools they develop themselves. They integrate CMVC successfully with other automated tools in their environments, molding it to the needs of their unique processes. Customers use CMVC as both a stand-alone product and as a product integrated into other software development environments, such as WorkBench/6000.

Perhaps most importantly, CMVC scales well—it handles linear, as well as lateral, growth in the requirements of the development organizations.

Appendix A. Customer Profiles

This appendix summarizes the profiles of the customers and the IBM locations we visited during the project that produced this redbook. It presents an overview of the environments in which CMVC is used and shows the characteristic features of each site (see Table 4).

Table 4. Customer Profile Matrix

	Continental	BT Laboratories	SCS	MCI	IBM Tucson	IBM Austin
Extent of use within company	Very small	Very small	Very small	Medium to large	Medium to large	Very large
Scale: • **Networking**	One LAN	One LAN	One LAN	Multiple LANs and WAN	Multiple LANs and WAN	Extreme use of networking
• **HW resources**	IBM 32H as CMVC server, 32H as DB server, 320 as CMVC clients	HP and SUN; SUN and IBM RISC System/6000 M360+M220/M250	IBM RISC System/6000 M580 as server, M340 as test, XStations and PCs	IBM RISC System/6000 M990 and M590 servers, several clients on several platforms	IBM RISC System/6000 M560 and M550 servers,several clients on several platforms	IBM RISC System/6000 2*M930, 2*M570, M950, M970, 2*M980 as servers, clients from AIX, OS/2, and VM
• **Number of users**	15	30	20	350	500	>3000
• **Number of projects/products**	3	6	2	20-30	several 100	several 1000
• **Number of families**	3	1+2	1	5	9	30
• **Number of components/files**	1 level hierarchy per family	4 level hierarchy per family	30 comps, 3000 files, 4 level hierarchy	100 comp/family, 30-50 files/comp	800 comps, 30000 files	largest family: 1800 components, 600,000 files
CMVC Version/Release	V 1.1	V 2.1 and V 2.2	V 2.2	V 2.2	V 1.1.2 and V 2.2	V 1.1 and V 2.2
Introduction into process	1993	1993	Mid 1993	1992	1992	since the pre-CMVC version was available

Appendix B. Software Configuration Management and Change Management

This appendix briefly describes the objectives of and advantages associated with the processes known as configuration management and change management. It also discusses the relationship between configuration management and change management and the relationship between them and other processes such as project management and software development methodologies.

The elements produced during the development of an application are created progressively, as new requirements are discovered and old ones are refined. One quickly loses track of the state of development of the application when individual elements were introduced. You can have the latest CASE tools and a highly skilled, 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 that 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 equally likely that some unauthorized fixes did manage to find their way into the application, creating a mismatch in interfaces, calls to nonexistent subroutines, or inappropriate access to data that no one can seem to explain. Problems of this sort represent failures in configuration management and change management.

As Figure 15 shows, simply having all of the right parts does not ensure a successful outcome in software development. It takes configuration management to ensure that the right parts are put together in the right manner, and change management to ensure that any changes to those parts or their relationships are well-thought out and deliberately applied.

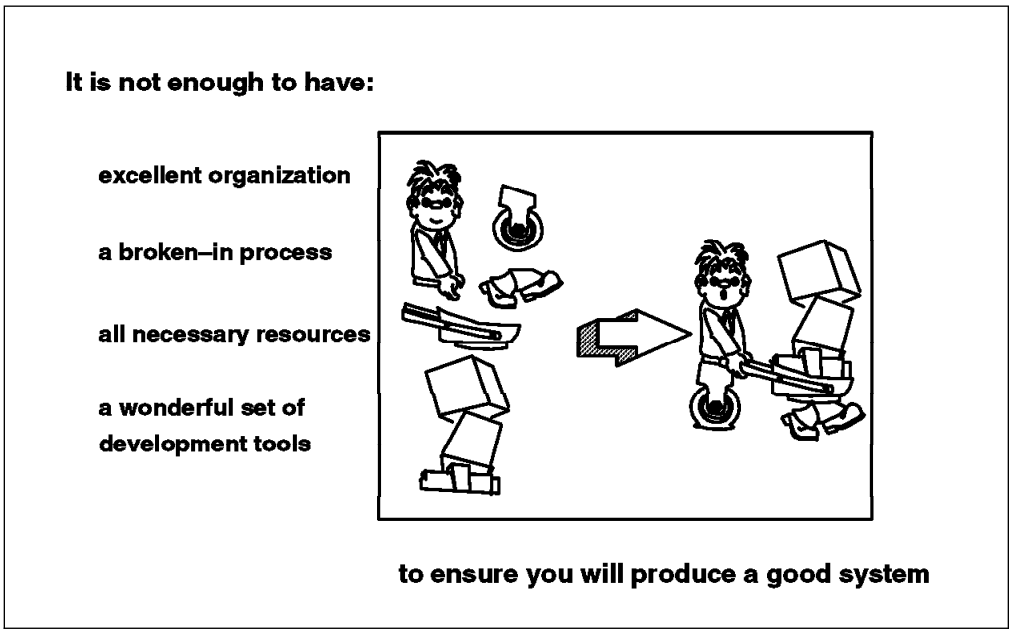


Figure 15. Why Configuration Management and Change Management Are Necessary

B.1 Why You Need Them

Configuration management and change management are put into place to prevent or cure problems that contribute to high development and maintenance costs, missed schedules, and customer dissatisfactions such as these:

- Impossibility of re-creating an exact previous version that exhibits the failing characteristic reported by end users
- Inability to trace changes in the application source code back to the specific enhancement in the functional requirements, or to an approved design change, or to a specific reported failure in the previous version of the application
- Unnecessary reworks, inconsistent integration test results, and general confusion as to who is working on which problems
- Degradation of the application between one release and another because untested components were inadvertently incorporated into the whole
- Incompatibility between two pieces of software sharing a single interface that has changed in only one of them
- Impossibility of upgrading the application by incorporating a series of changes in a controlled and validated sequence
- Inability to generate different parallel versions of the same application because the parts that are unique to each version cannot be adequately identified
- Inability to report on the exact status of development or maintenance tasks
- Inability to identify exactly which developer is charged with implementing a particular change to the application.

B.2 The Goals

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

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

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

Another very important function is to identify precisely 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
- Site-specific or platform-specific release.

Identifying all changes to a given baseline and ensuring that they are incorporated into the next newly forming baseline in an orderly and controlled manner are the core responsibilities of change management. Change control identifies and tracks problems and suggested enhancements to an application, thereby ensuring that

each is carefully evaluated, and—if approved— correctly implemented and incorporated into the application.

B.3 The Formal Definition

The classic definition of configuration management is given by the U.S. Department of Defense in its standard on software development, DOD-STD-2167A:

"Configuration management is the discipline of identifying the configuration of software systems at discrete points in time for the purpose of controlling changes and maintaining traceability of changes throughout the software system life cycle."

Configuration management is often divided into subprocesses:

- Configuration element identification: establishes a precise nomenclature for all configuration elements, such as files, subsystems, and systems and provides a unique identifier for each of them
- Configuration control: manages the production of all elements of the application and the integration of those elements into a complete configuration
- Change control: records and tracks all change requests through their ultimate disposition
- Configuration audits: compares one baseline against its preceding baseline to ensure consistency between the two.

B.4 What They Do

Changes to the application baselines occur continuously throughout the software life cycle. Configuration management and change management provide mechanisms to:

- Define and identify all elements of the application
- Define and identify how those elements are combined to build the complete application
- Keep track of change requests and their status (approved, assigned, implemented, tested, incorporated into the application)
- Maintain the traceability between all changes to the application and the change requests whose approval authorized those changes
- Retrieve a given previously established baseline
- Retrieve all changes applied between one baseline and the next subsequent baseline
- Allow gradual incorporation of sets of changes into the developing baseline, and deincorporation of a set of changes that causes failure during integration testing.

The processes associated with configuration management and change management ensure orderly development of software and enable a development history to be created. This history provides traceability and makes it possible to perform audits. It also enables statistics to be produced in order to evaluate the impact that an update may have on the software currently being developed or to be developed.

B.5 Who Needs Them

Any application development effort worth doing is worthy of configuration management and change management. This is usually self-evident in the case of big and medium-sized application development or software engineering efforts, but it is not always clear from the start of smaller efforts. Whether the need is from the first step of a project or better placed near the end of a project may depend on the size and complexity of the effort and the application. How complete and strict the procedures are that govern configuration management and change management may be determined by the size of the investment, the risk/benefit ratio, and company- or industry-imposed standards. But configuration management and change management are critical to the success of any major development effort, and they are cost-effective even during the maintenance phase of smaller efforts.

B.5.1 Big Development Efforts

Consider a large software development effort such as that of developing and maintaining a major operating system. With a reasonable assortment of application products and operating system code, a typical UNIX software system might take up 400MB on disk when installed. It would support dozens of types of hardware peripherals, contain hundreds of end-user commands, include dozens of libraries and APIs, and provide a handful of compilers. At any one time, it probably has a half-dozen releases in the field and runs on at least three or four hardware architectures. The company selling such an operating system might employ hundreds of developers, testers, documenters, and quality assurance people to handle this job.

Such a company could not risk the loss of profit and sales caused by any confusion in the generation, maintenance, or delivery of its operating system and related software products. Communications among so many people could not be managed by word of mouth or random electronic communications. Project management would need sophisticated methods of measuring and auditing the development process. Quality assurance would have to be built in to every step of the development process. Clearly this company could not begin to manage an effort of this scope without very strict and widely encompassing configuration management and change management procedures and policies and significant automated tool support.

B.5.2 Medium-Sized Development Efforts

Now, consider a smaller business application development effort that handles customer, order, and payment information for products shipped on customer subscription. The application has COBOL, C, and C++ components, and some of the source code is automatically generated by a 4GL type of tool. It executes in parallel on two platforms, AIX and MVS, and will be implemented on AIX in two phases. It has a client/server architecture and is built on a commercial relational database. It has a nongraphical user interface on the MVS mainframe, and a GUI on AIX. It consists of a few dozen source files for each version on each platform, several build instruction files, some database instructions to create and populate the tables, and some text source files for end-user documents. This project will require only three or four developers to implement the AIX version and maintain the MVS version during that effort. It will have a project manager, some quality assurance oversight, and some end-user testing support.

This project will also require configuration management and change management. The few people involved in this project will not be able to modify and maintain multiple versions of the source files, readily identify the equivalent user interface files for each platform, and ensure that common code for both platforms is compatible with both compilers by keeping short-hand notes and sending each other occasional electronic mail. They will not be able to use simple mechanisms as separate directories and file naming conventions to ensure that use of common code is maximized while platform-specific and release-specific code is fetched properly during the product build process. In no time at all, this simple project will get out of hand.

B.5.3 Small Development Efforts

It is often necessary to convince software developers to apply configuration management and change management to smaller projects. The value of these disciplines to smaller development efforts is often underestimated. This type of management is perceived to require a level of effort and formality that is excessive compared to the total lines of code or the number of developers required to implement a small application. Developers also incorrectly associate configuration management and change management with a restraint on their creativity or an impediment to their rapid progress.

However, even an application that is so small that it does not require formal design or independent testing still requires configuration management and change management during its maintenance phase. Often, a company will have a whole assortment of small applications that collectively constitute a significant investment in development effort.

If even one end-user's ability to do his or her work comes to depend on the availability and functionality of a small application, maintenance will be necessary some day. As time passes, the environment in which an application executes will change, some external interface will change, or the set of requirements met by the application will change. In each case, program maintenance will be required to solve problems resulting from these changes.

Application maintenance would be impossible if the application source and instructions to rebuild it have not been tightly controlled since the application first went into production use.

B.6 Interaction with Development Methodologies

Source code is only one expression of the application. That type of objects that constitute a preliminary development baseline will depend on the development methodology chosen by the project. The milestone events and the types of baselines may vary, but the principles of configuration management and change management will not.

If a project applies the traditional waterfall methodology, the baselines controlled might be:

- Functional and interface requirements
- Build-to system, subsystem, and component designs
- As-built test, integration, and delivery implementations.

In the waterfall case the objects managed by configuration management and change management might be files containing:

- Various requirements specifications
- Various forms of design notation and data definitions
- Source and executable application code.

If a more recent methodology, such as the Grady Booch object-oriented methodology is used, the baselines might be:

- Conceptualization prototype
- Analysis description that models the behavior of the application
- Architectural release and descriptions of tactical policies
- Successively refined executable releases.

The objects controlled by configuration management and change management in the object-oriented case might be files containing:

- Object and class diagrams, finite state machines, documented nonbehavioral aspects of the design such as portability, reliability, security, and efficiency
- Class and object structure diagrams and an architectural release
- Source and executable application code.

Configuration management and change management procedures and mechanisms must be tailored to meet the requirements of the development methodology.

B.7 Interaction with Project Management

Configuration management and change management provide input and assistance to project management. For example, an upgrade to the application may imply a change in the set of defined activities or milestones. The investigation performed while evaluating the impact of a given change will alert project management of a need to make changes in the development schedule or apply additional human resources to the project.

Configuration management and change management mechanisms can be used to ensure consistency across the application in such many ways as checking the presence and content of module headers, recording approval of quality assurance representatives, ensuring related design updates, test results, software quality metrics data and that upgraded end-user documentation are submitted with related code changes.

Change management mechanisms provide for input from various members of a project including management itself. Change management provides a means by which management can assign responsibility for the implementation of changes as well as monitor their progress.

B.8 Interaction with Quality Assurance

Configuration management and change management implement the specific policies and practices espoused and adopted by a development effort. Quality assurance organizations exist to inspect and verify that the effort conforms to these as well as to any externally applied rules and regulations. Quality assurance oversight of the application development process is very much enhanced by formal configuration management and change management procedures. These procedures can be implemented in such a way as to ensure that software does not

enter a baseline unless it has been approved by quality assurance representatives and is accompanied by the required supporting material, such as test plans and test results. These representations can ensure that changes to a formal baseline are always accompanied by identification of the corresponding problem report. They can ensure that software conforms to project quality standards regarding such topics as module headers, minimum number of lines of comments, and naming conventions. Quality assurance representatives can easily inspect and approve these procedures and the resulting data.

B.9 Configuration Management History and Statistics

If configuration management and change management are implemented in connection with a database providing generalized query capability, risk, cost, quality metrics, and other data can be captured and analyzed for project management uses also. Various statistics related to project management, software quality configuration management, and change management can be derived from a well-maintained configuration management database. It is possible to extract sufficient data from such a database to project cost, staffing, software sizing, and development schedule data of similar projects under planning.

Figure 16 shows the relationships between configuration management, change management, and the application development environment.

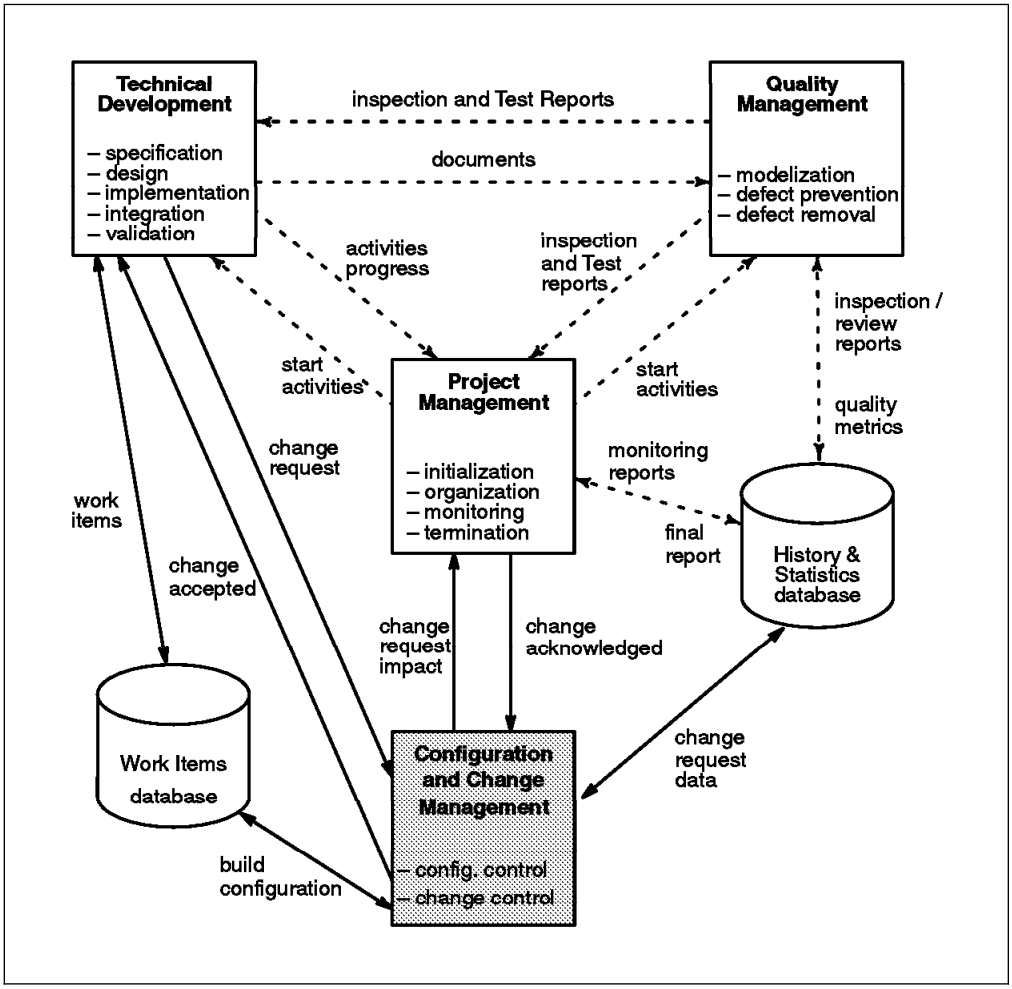


Figure 16. Development Process Relationships

B.10 CMVC Automated Support

for Configuration Management and Change Management While procedures and practices to implement configuration management and change control can be manual, and in fact were for many years, these disciplines lend themselves particularly well to automation. With the advent of relational database technology, data about the configuration objects and change reports could be accumulated and accessed in a variety of ways if coherently maintained by automated configuration management and change management software. As application development became increasingly complex, it became not only more convenient, but absolutely necessary to automate most of the tasks and procedures supporting configuration management and change management.

When IBM undertook to develop its own UNIX-based operating system, AIX, it discovered it needed a RISC System/6000-based configuration management and change management system of industrial strength. The AIX development effort was on the same scale as the sample big development effort described earlier. After looking over the marketplace, IBM decided to develop its own system. No one product at that time included all of the features that IBM knew it required for AIX development.

The original product developed for internal use was called Orbit. IBM soon realized that if Orbit could support the AIX development effort, other software engineering and business application developers could also benefit from it, so the decision was made to bring Orbit out as the product known as CMVC/6000.

One benefit of using CMVC is that it helps to establish a software development process within the organization. The process is defined and enforced by CMVC.

CMVC makes the enforcement of the process easy for the project leaders, because they can rest assured that none of the required procedural steps is bypassed. CMVC also makes the following of the process easy for the software developers themselves, by automatically notifying each person that must take some action or provide some input to the system. Mistakes and wasted time are therefore eliminated when everyone that is part of the process knows exactly what to do and when. Managers are kept informed of the state of the project, and of the changes occurring with the product under development. The software developers are informed when they have been assigned a task (for example, fixing a bug or implementing a feature) and are automatically given the proper access to the source files that need to be modified.

In addition, CMVC provides an audit trail of all changes to each of the files, as they relate to each defect fixed or feature implemented. Such an audit trail makes it easy to pinpoint the modules that have been the most error-prone or that were modified most for features. Such modules can then be subjected to close scrutiny to eliminate chances of defects being introduced into the product. Metrics relating to defects and features are being kept within CVMC, and such metrics provide valuable insight into the quality of the product.

B.10.1 Increase in User Productivity

The CMVC actions that a developer must take while fixing a defect or implementing a feature are the steps that he or she normally takes; therefore no extra effort is necessary when using CMVC. Having problem tracking functions integrated with version control and change management will actually save time and effort. A developer will not have to work with two or three separate systems to work with and keep in sync. The information, data, and access to the source code is at his or her fingertips, available through an easy-to-use, intuitive GUI.

Recognizing that today's software development organizations will develop software for different platforms, CMVC provides a consistent user interface across all major UNIX platforms: RISC System/6000, Sun SPARCstations and HP Apollo 9000 workstations, and OS/2 and DOS Windows. Therefore, a developer moving from one workstation to another will see no difference in the way he or she uses CMVC.

To reduce the costs associated with maintaining the software product once it has been released, CMVC allows users to extract and build exact duplicates of the software that was shipped some time ago. Also, development work is allowed from that level of the software onward, for example, if a bug is reported for version 1.1 of a product and the current working version is 1.3, the bug can be fixed at both the 1.1 level and the 1.3 level simultaneously. The customer, in this case, will receive a fix applicable to the version of the product that it is using (1.1 in this example). To further reduce wasted time spent understanding older changes to the code, CMVC provides the change history at both a low level (file-specific) and a high level (conceptual, for example, a defect being fixed or a feature being implemented). A developer will therefore be able to find out to which group of changes (defined as a defect or a feature) an individual file change belongs.

Automatic Documentation Generation for Change: The automatic documentation allows developers to add their own comments separate from the source files being worked on. In addition, the entire workgroup is kept informed of the changes made and the comments entered.

Through the integration of the problem tracking mechanism and the change tracking mechanism, CMVC will automatically provide change-specific documentation that answers the who, when, why, and what questions.

In addition, users can enter comments at any time to clarify the changes that are being made. All comments entered by users during the defect-fix cycle and the feature-implementation cycle are saved in the database and sent to all interested members of the workgroup. The communication within the workgroup is therefore enhanced.

Investment Protection in Hardware and Software: By supporting all major OS platforms (UNIX, OS/2, DOS Windows), CMVC allows you to keep your current investment in the heterogeneous hardware environment that your organization currently owns.

A migration path is provided for source code currently under development using UNIX SCCS.

The command-line-based CMVC interface to a graphical CMVC interface to integrated CMVC operation through SDE WorkBench or SoftBench is made without

any loss or modification of data. All three modes of access can be used concurrently, depending on which setup each individual user has.

Scalability According to Your Organization: CMVC is suited for both small, tight workgroups and large, multisite organizations working on common projects. There is no practical limit to the number of users that CMVC can simultaneously support, the real limits being imposed by hardware constraints. The client/server model of CMVC is well suited for growth and expansion.

Customized Report Facility: The Report function of CMVC is one of the most powerful in the industry, allowing users to gain access to any of the historical data, metrics, project status in any form desired. CMVC inherits its powerful reporting capabilities from the underlying relational database.

Reuse Facility of Software Parts: It is very easy to share source code when the source code you want is managed by CMVC. One simply has to include the files to be shared in the current release of the product. The shared modules will remain shared (therefore will evolve the same way) for as long as you want. If you decide to keep only part of them shared while modifying some for the private needs of your project, CMVC allows you to do this with a minimum of wasted storage space.

CMVC for Everybody in Your Organization: CMVC allows the definition of user roles. Each role is defined according to the actions it can take. For example, a workgroup may decide to have seven roles defined: manager, project leader, team leader, developer, tester, builder, and general user. Any one CMVC user must assume at least one role but can also have multiple roles (typical in smaller organizations). Also, a user can have different roles in different projects, or in different parts of the same projects. Actually, there is no restriction on the roles one person can have anywhere within the organization.

Access control is based on user roles and extends well beyond simple read-write-execute permissions that other so-called CM systems have.

Many Other Functional Benefits

- Project complexity is reduced by the creation of manageable components.
- Design/defect management is integrated with configuration management and with version control.
- Source changes are tracked and controlled across multiple releases, components, and environments.
- Source repository is centralized while access is distributed, using a client/server model.

Appendix C. Implementation of ISO 9001 Using CMVC

This appendix provides a brief introduction to the International Standard Organization (ISO) 9000 and describes how the IBM CMVC tool can be utilized to meet some key ISO 9001 elements.

The software engineering industry is the fastest growing industry in the last half of the century. Throughout the industry, software development organizations are struggling with the challenges of reducing costs, increasing productivity, and improving quality. Toward these efforts, quality management of software is essential. One way to establish a quality management system is to provide guidance for software quality assurance. Such guidance is found in the ISO 9000 series of quality standards: ISO 9001, ISO 9002, ISO 9003.

ISO 9000 compliance is of key importance to organizations if they are to survive the fierce competition of the 1990s and beyond. With the introduction of ISO 9000, the software engineering industry has experienced a shift toward implementing techniques and processes aimed at developing processes that are well defined and repeatable. Adoption of these proven techniques and processes allows an organization to improve the overall process of:

- Creating world class software
- Maintaining a dynamic, responsive, and innovative environment
- Attaining a high return on investment through the pursuit of total customer satisfaction.

Software tools can assist in improving an organization's management system and meeting ISO 9000 compliance. CMVC is an effective tool that simplifies the organization and management of diverse tasks involved in software development so that you can improve your entire product development process and it can be used to comply to some key elements of ISO 9001.[2]

C.1 ISO 9000

To facilitate the standardization of the many aspects of quality, the ISO has developed a set of international standards for quality systems which are known as the ISO 9000 Series of Quality Standards. These standards apply to all organizations producing a product or service and are being accepted world-wide. An indication of their acceptance and significance in the software engineering industry, in particular, is reflected in Hübner's words, “To obtain an ISO 9000 certification has become a business necessity in Europe” [1]. It is only a matter of time before ISO 9000 certification becomes a business necessity in North America and the Orient.

Three key ISO 9000 standards are:

- ISO 9001, Quality systems - Model for quality assurance in design/development production, installation and servicing.

[2] IBM software development Labs in Toronto and Austin use IBM CMVC to manage and control the software development process and implement certain elements of the ISO 9001 standard.

- ISO 9002, Quality systems - Model for quality assurance in production and installation.
- ISO 9003, Quality systems - Model for quality assurance in final inspection and test.

ISO 9001 provides the most comprehensive requirements for a software quality system where a contractual agreement between two parties must demonstrate the supplier's ability to design and supply a product or service.

Recognizing the peculiarities of the software industry, ISO 9000-3, a guideline for the application of ISO 9001 to the development, distribution and maintenance of software, has been released. Configuration management is a key element in this ISO 9000-3 guideline for software.

C.2 CMVC and ISO 9001

Configuration management provides a mechanism for identifying, controlling and tracking the versions of each software item. In many cases, multiple versions of software items are in use and must be maintained and controlled. Configuration management itself is not an element of the ISO 9001 standard (only an element of ISO 9000-3 guidelines); however, the following elements of ISO 9001 depend on configuration management:

- Document Control
- Design Control
- Product Identification and Traceability
- Inspection and Test Status
- Control of Nonconforming Product
- Internal Quality Audits.

Only certain aspects of Document Control, Design Control, Control of Nonconforming Product and Internal Quality Audits are addressed by Configuration Management. Product Identification and Traceability and Inspection and Test Status are fully addressed by it.

The remaining sections describe these ISO 9001 elements and highlight how CMVC can support them.

C.2.1 Document Control

Document Control covers:

- The determination of those documents that should be subject to document control procedures
- The approval and issuance of document control procedures
- The change procedures including withdrawal and, as appropriate, release. [3]

The aspects of Document Control that can be successfully addressed by CMVC are:

- Documents must be accessible to a group of people with predetermined interest and authority.

- The changes to the content of a document under document control have to be reviewed by a prespecified group of reviewers and the final version of the document has to be approved by them.
- All the users of the document have to be notified of the changes to it.
- Once the new document is finalized and becomes the most recent working document, provisions must be taken to prevent users from using a back-level version of the same document.

Files and documents pertaining to a particular project reside in one more CMVC components (where each CMVC component is dedicated to a specific department and/or all documents related to a particular project).

An access list and a notification list is associated with each CMVC component. The type of access each user has to the documents stored in CMVC depends on the user's role in the development team. The type of notification each user has depends on the interest or need to be informed of changes to documents in the CMVC environment. Access authority and notification subscription is assigned to a user by the component owner or by someone who has the authority to grant other users access and notification to a specific component. When a document is updated, the owner of the managing component where the document resides and all users who have subscribed to being informed of document updates receive mail notifying them of the update.

When the CMVC approval process is activated, approval must be given for proposed changes before work can begin on the implementation of a change. Approvers specified for each release need to review the information recorded in the defect or feature and evaluate the proposed changes to the release in relation to other project considerations. A CMVC approval record is created for each approver. Each approver indicates his or her evaluation of the changes and can optionally append comments to the defect or feature to explain the rationale for his or her decision. File changes for that defect or feature in that release cannot be checked in to the CMVC server until all approvers accept the proposed changes. [4]

C.2.2 Version Control in ISO 9001

Certain aspects of Design Control, Product Identification and Traceability, Inspection and Test Status, and Control of Nonconforming Product deal with the issue of *version control*. Activities are versioning documents and source modules that compose a product; identifying the various versions of the documents and source modules and the reasons why changes were made from one version to the next; inspecting and testing the content of each version and recording the status of this outcome; and finally, controlling nonconforming products and identifying the version of documents and source modules that reflect the nonconformance.

Version control, by definition, is the storage of multiple versions of a single file along with information about each version. [4] CMVC provides for version control and enhances this basic function with an extra layer of traceability so that each version is cross-referenced to a reported defect or a suggested enhancement.

The following sections highlight the specific sections of each of the ISO 9001 elements that emphasize the need for version control mechanisms in an organization.

C.2.2.1 Design Control

The ISO 9001 element of Design Control states that “the supplier shall establish and maintain procedures to control and verify the design of the product in order to ensure that the specified requirements are met.” One of the items that define this element of Design Control is the Control of the Design Changes, which is designed as:

> **“**The supplier shall establish and maintain procedures for the identification, documentation and appropriate review and approval of all changes and modifications**.”** [2]

CMVC can be used to control and verify the design of a product in a number of ways. First, the design specification documents themselves can be stored in CMVC. Modifications to the content of the design specifications can be controlled from both an access and an update perspective. Development teams can make use of the problem-tracking feature of CMVC to track and control the changes to design specifications and to ensure that all changes have been well documented, justified, and reviewed for appropriateness and applicability. CMVC's design, size and review process for reported defects and suggested features provides for the identification, documentation and appropriate review of all changes and modifications. CMVC's tracking process allows development teams to cross reference changes to design documents or source modules to the reported defects and features. It also provides an additional layer of control to those teams seeking an approval checkpoint prior to versioning the documents and a review of the changes after making the modifications but prior to committing them.

When source modules for a product are managed by CMVC, development teams can ensure that the changes made to the product are consistent with its design by using the integrated problem tracking and change control feature of CMVC. Attributes of reported defects and features can be used to cross-reference design documents with the proposed source module changes to the product. CMVC's ability to track required changes in all project deliverables ensures that appropriate updates are made to design documents, source modules, test cases and end-user documentation for each reported defect and suggested enhancement of the product.

When the time comes to release a product, development teams can benefit from CMVC's ability to maintain multiple versions or variants of a product. Enhancements for the next release can be incorporated into design documents and source modules without disrupting the integrity of the products that have been distributed.

C.2.2.2 Product Identification and Traceability

The ISO 9001 element applicable to version control for Product Identification and Traceability states:

> **“**here appropriate the supplier shall establish and maintain procedures for identifying the product from applicable drawings, specifications or other documents, during all stages of production, delivery and installation. Where and to the extent that, traceability is a specified requirement, individual product or batches shall have a unique identification. This identification shall be recorded.**”**[2]

When CMVC is used as the configuration management and version control tool for software development activities, several levels of identification and traceability are available.

Each version of a document or source module is identified by a version number. The combination of this version number, as well as the document or source module name (CMVC file name), and the product name that the document or source module is associated with (CMVC release name), uniquely identify a file in CMVC.

As previously discussed, changes to documents and source modules can be cross-referenced to CMVC defects and features. Users can then query the history of changes and identify the content of each version of a document or source module and the reason why it has changed over time. Alternatively, users can query the details of the defects and features to determine the document or source modules that were changed as a result of fixing a reported problem or implementing a suggested feature.

CMVC records the time and date of the changes to documents or source modules as well as the user who makes each change. This information provides additional traceability and can be queried at any time.

C.2.2.3 Inspection and Test Status

The ISO 9001 element applicable to version control for Inspection and Test Status states:

> "...The identification of inspection and test status shall be maintained, as necessary, throughout production and installation of the product to ensure that only product that has passed the required inspections and tests is dispatched, used or installed. Records shall identify the inspection authority responsible for the release of conforming product."[2]

The CMVC problem tracking mechanism allows development teams to establish two types of testing procedures. When development teams use the tracking mechanism, they can define test environments and testers for each release of a product. As defects are fixed and features are implemented, CMVC activates test records for each of the test environments and testers indicating when the changes made to documents and source modules have been committed in the product. Testers are notified when their test records are ready to be marked. By marking a test record with an accept, reject, or abstain status, a tester relays information to the development team as to the status of the change. When test records are rejected, additional defects or features can be opened to track the nonconformances, and attributes in both the original and the new defects or features can be used to cross-reference problems of a similar nature.

Another type of testing occurs at the end of the CMVC defect or feature lifecycle. Once applicable documents or source modules have been changed and committed into the various products, the originator of the defect or feature has the opportunity to verify that the resolution of the problem or the implementation of the suggestion has been accomplished to his or her satisfaction. Originators record their satisfaction of the outcome on verification records.

The status of test and verification records, the owner of test and verification records, and the timestamp of when each record was last updated is maintained in

CMVC. The reporting mechanism allows users to query the status of these records at any time.

C.2.2.4 Control of Nonconforming Product

The ISO 9001 element applicable to version control for the Control of Nonconforming Product states:

> "....Control shall provide for identification, documentation, evaluation, segregation (when practical), disposition of nonconforming product and for notification to the functions concerned." [2]

Nonconformances with respect to products managed by CMVC are identified by opening a CMVC defect or a CMVC feature. Once the nonconformances are identified, the development teams can evaluate the validity of the nonconformance, and can return the defect or feature as invalid, as a documented deviation, or as a nonconformance that has already been addressed by another defect or feature report. Alternatively, the development team can decide to accept responsibility for the nonconformance and schedule its resolution in the appropriate product releases. In either case, all users who have subscribed to defect and feature reports and state changes will be kept informed.

When nonconformances are received for products that have been released to customers, development teams can resolve the nonconformance and reissue a product update. Development teams can make use of the attributes associated with product levels to describe the status of the package. For instance, a product level may be shipped and then subsequently replaced with a newer version that includes fixes for nonconformances.

C.2.3 Internal Quality Audits

> ".....The audits and follow-up actions shall be carried out in accordance with documented procedures. The results of the audits shall be documented and brought to the attention of the personnel having responsibility in the area audited. The management personnel responsible for the area shall take timely corrective action on the deficiencies found by the audit." [2]

An example of how IBM addressed this element using CMVC, is the internal audits to check the level of compliance of the various departments in the IBM PRGS Toronto Lab. Internal audits were conducted and nonconformances were issued and monitored until a satisfactory corrective action plan was put in place and successfully implemented.

In a specific area, a component dedicated to ISO 9001 was created and an owner was assigned to it. Traditionally, the owner of this component is the ISO focal point for the area. Each department in the area had its own component where all the documents and processes were stored.

The internal audit group would open defects (Nonconformances) against the area component. The component owner would then route the nonconformances to the corresponding department component. The departments are responsible for creating their own corrective action plan. Each corrective action was appended to the nonconformance in CMVC, and all the interested parties were notified that remarks were appended to the specific nonconformance.

The remarks added to each of the nonconformances that described the corrective action were retrieved via CMVC and then forwarded to the internal audit group for review. This group created the actual corrective action plan for each department and hence for the area as a whole.

C.3 Conclusion

To make software quality a reality and to comply to the ISO 9001 standard, CMVC can successfully be used for:

- Design Control
- Document Control
- Product Identification and Traceability
- Inspection and Test Status
- Control of Nonconforming Product
- Internal Quality Audits.

C.4 Brief Description of ISO 9000-3

This section describes the elements listed in the section C.2, "CMVC and ISO 9001" on page 114, as well as the 9000-3 element on Configuration Management.

C.4.1 Configuration Management

Configuration Management should:

- Uniquely identify the versions of each software item
- Identify the versions of each software item that together constitute a specific version of a complete product
- Identify the build status of software products in development or the status of those software products delivered and installed
- Control simultaneous updating of a given software item by more than one person
- Provide coordination for the updating of multiple products in one or more locations as required
- Identify and track all actions and changes resulting from a change request, from initiation through to release.

C.4.1.1 Configuration Identification and Traceability

To establish and maintain procedures for identifying software items during all phases, starting from specification through development, replication and delivery, each individual software item should have a unique identification.

There should be provisions to uniquely identify the following items for each version of the software:

- The functional and technical specifications
- All development tools which affect the functional and technical specifications
- All interfaces to other software and/or hardware items
- All documents and computer files related to the software item.

The identification of a software item should be handled in such a way that the relationship between the item and the contract requirements can be demonstrated.

For released products, there should be procedures to facilitate traceability of the software item or product.

C.4.1.2 Change Control

Procedures should be in place to identify, review and authorize any changes to the software items under the control of configuration management. All changes to software items should be carried out in accordance with these procedures.

Before a change is accepted, its validity should be confirmed and the effects on other items should be identified and examined.

Methods to notify those concerned of the changes and to show the traceability between changes and modified parts of software items should be provided.

C.4.1.3 Configuration Status Report

The supplier should establish and maintain procedures to record, manage and report on the status of software items, of change requests and of the implementation of approved changes.

C.4.2 Design Control

The supplier should establish and maintain procedures to control and verify the design of the product in order to ensure that the specified requirements are met.

C.4.3 Document Control

Procedures should be established and maintained to control all documents that relate to the contents of this part of ISO 9000. They cover:

1. The determination of those documents that should be subject to the document control procedures.
2. The approval and issuance of document control procedures.

   All documents should be reviewed and approved by authorized personnel prior to issue. Procedures should exist to ensure that the pertinent issues of appropriate documents are available at appropriate locations where operations essential to the effective functioning of the quality system are performed. Obsolete documents should be promptly removed from appropriate points of issue or use.

   Where use is made of computer files, special attention should be paid to appropriate approval, access, distribution and archiving procedures.
3. The change procedures including withdrawal and, as appropriate, release.

C.5 References

1. Hübner, Achim *ISO 9000 Implementation in Germany* , ***LOGON***, Volume 4, Number 4, September 1992 (IBM Internal Use)
2. *International Standard: ISO 9001*, Reference Number ISO 9001:1987(E)
3. *International Standard: ISO 9000-3*, Reference Number ISO 9000-3:1991(E)
4. *IBM CMVC Concepts*, SC09-1633-00, IBM Corporation, 1993.

Glossary

absolute path name. A directory or a file expressed as a sequence of directories followed by a file name beginning from the root directory.

access list. A CMVC object that controls access to development data. A list of user ID-authority group pairs attached to a component, designating users and the corresponding authority access they are being granted for all objects managed by this component or any of its descendants. It also contains the user ID-authority group pairs designating users who are restricted from performing actions at a specific component.

action. A task performed by the CMVC server and requested by a CMVC client. A CMVC action corresponds to issuing one CMVC command.

approver. A user who approves changes within a specific release.

approver list. A list of user IDs attached to a release, representing the users who must approve file changes required to resolve a defect or implement a feature in that release.

ASCII. The standard coded character set using 7-bit characters (8th bit for parity) used widely on non-IBM mainframe computers.

authority. The right to access development objects and perform CMVC commands. See also *access list*, *base authority*, *explicit authority*, *implicit authority*, *restricted authority*, and *superuser privilege*.

base authority. The set of actions granted to a user whenever a user ID is created within a CMVC family.

base file name. The name assigned to the file outside the CMVC server environment, excluding any directory names.

batch program. A batch program reads its input from a file or device and writes its output to a file or device without the interaction of a user.

change control. The process of limiting and auditing changes to files through the mechanism of checking files in and out of a central, controlled storage location. Change control for an individual release can be integrated with problem tracking by specifying a process for that release that includes the track subprocess.

check in. The return of a CMVC file to version control.

check out. The retrieval of a revision of a CMVC file from version control.

child component. All components in each CMVC family, with the exception of the root component, must be created in reference to an existing component. The existing component is referred to as the parent component, while the new component becomes known as the child component. A parent component can have more than one child component. See also *component*.

client. A workstation that requests services from another workstation.

command. A request to perform an operation or run a program from the command line interface. In CMVC, a command consists of the command name, one action flag, and zero or more attribute flags.

common file. A file that is contained in two or more releases and the same version of the file is the current version for those releases.

component. A CMVC object that simplifies project management, organizes project data into structured groups, and controls configuration management properties. Component owners can control access to development data (see *access list*) and configure notification about CMVC actions (see *notification list*). Components exist in a parent-child hierarchy, with descendant components inheriting access and notification information from ancestor components.

configuration management. The process of identifying, managing, and controlling software modules as they change over time.

context. A description of a data file or directory in the form host dir file. That is the host machine, working directory, and file.

database. A systematized collection of data that can be accessed and operated upon by a data processing system for a specific purpose.

defect. A CMVC object used to formally report a problem. The user who opens a defect is the defect originator.

delete. Deleting a development object, such as a file or a user ID, within CMVC. Certain objects can be deleted only if certain criteria are met. Most objects that are deleted can be re-created.

destroy. The only CMVC development object that can be destroyed in CMVC is a file. Destroying a file removes the file record from the database on the CMVC server. Though a destroyed file cannot be re-created, it will appear as part of an extracted level.

EBCDIC. A coded character set of 256 8-bit characters used on IBM and other mainframes.

end user. See *user*.

environment. A user-defined testing domain for a particular release. Also used as a defect field, in which case it is the environment where the problem occurred.

environment list. A CMVC object used to specify environments in which a release should be tested. A list of environment-user ID pairs attached to a release, representing the user responsible for testing each environment. Only one tester can be identified per environment.

explicit authority. The ability to perform an action against a CMVC object because you have been granted the authority to perform that action.

extract. A CMVC action you can perform on a file, level, or release. A file extraction results in the specified file being copied to the client workstation. Level extraction and release extraction result in copying the files associated with the level or release to a designated workstation.

family. A logical organization of related development data. A single CMVC server can support multiple families. The data in one family cannot be accessed from another family.

family administrator. A user who is responsible for all non-system-related tasks for one or more CMVC families such as planning, configuring, and maintaining the CMVC environment and managing user access to those families.

feature. A CMVC object used to formally request a functional addition or enhancement. The user who opens a feature is the feature originator.

file. A collection of data that is stored by the CMVC server and retrieved by a path name. Any text or binary file used in a development project can be created as a CMVC file. For example, source code, executable programs, documentation, or test cases.

fix record. A status record that is associated with a track and is used to monitor the phases of change within each component that is affected by a defect or feature for a specific release.

function key. A key appearing at, above, or beside the normal character keys on a keyboard which can be programmed to perform particular functions in particular program contexts.

home directory. The directory users access when they log in.

host. Host node, host computer, or host system.

host list. A list associated with each CMVC user ID which indicates the client hosts that can access the CMVC server and act on behalf of the CMVC user. The list is used by the CMVC server to authenticate the identity of a CMVC client upon receipt of a CMVC command. Each entry consists of a login, a CMVC user ID, and a host name.

implicit authority. The ability to perform an action against a CMVC object without being granted explicit authority. This authority is implicitly granted due to object ownership. Contrast with explicit authority and base authority.

inheritance. The passing of configuration management properties from parent component to child component. The configuration management properties that are inherited are access and notification. Inheritance within a component hierarchy is cumulative.

Korn shell. The default UNIX shell executed on AIX. It is virtually identical to the proposed POSIX standard shell.

level. A collection of tracks which represent a set of changed files within a release.

level member. A track that has been added to a level.

lock. Prevent editing access to a file stored within the CMVC development environment so that only one user can make changes to a given file at one time.

login. Operating system user identification.

make. The make command assists you in maintaining a set of programs, usually pertaining to a particular software project. It does this by building up-to-date versions of programs.

makefile. This description file tells the **make** command how to build the target file, which files are involved, and what their relationships are to the other files in the procedure.

NEDS. A configuration management tool developed and used internally within IBM, which was used as the base for developing CMVC.

Network File System (NFS). A program that allows you to share files with other computers in one or more networks over a variety of machine types and operating systems.

NFS. Network File System

notification list. A CMVC object allowing component owners to configure notification. A list of user ID-interest group pairs attached to a component,

designating users and the corresponding notification interest they are being granted for all objects managed by this component or any of its descendants.

online program. A user provides input interactively to an online program and views its output on a display, panel, or window.

OPATS. A configuration management tool developed and used internally within IBM, which was used as the base for developing CMVC.

Open Systems. Operating systems which are available on many different vendors' computers, across which programs may be easily ported. UNIX and DOS, two operating systems which are available on a great many vendor platforms, are both considered open systems by many people. Typically, an open system supports *de facto* industry and *de jure* formal standard interfaces, subsystems, languages, and utilities.

originator. The user who opens a defect or feature and is responsible for verifying the outcome of the defect or feature on a verification record. This responsibility can be reassigned.

owner. The user who is responsible for a CMVC object within a CMVC family, either because they created the object or because they were assigned ownership of that object

parent component. See *child component* and *component*.

path name. The name of the file under CMVC control. A path name can be a set of directory names and a base name or just a base name. It must be unique within the release that groups the files.

PLIST. A set of tools that were used internally within IBM in conjunction with OPATS.

Orbit. A predecessor tool of CMVC that was used internally within IBM

problem tracking. The process of tracking all reported defects through to resolution and proposed features through to implementation.

profile. A file containing customized settings for a system or user.

process. A combination of CMVC subprocesses, configured by the family administrator, that controls the general movement of CMVC objects (defects, features, tracks and levels) from state to state within a component or release. See also *subprocess* and *state*.

query. A structure request for information from a database, for example, a search for all defects that are in the open state. See also *search*.

relative path name. The name of a directory or a file expressed as a sequence of directories followed by a file name, beginning from the current directory.

release. A CMVC object defined by a user to group all files that must be built, tested, and distributed as a single entity.

restricted authority. The restriction of a user's ability to perform certain actions at a specific component.

root component. The initial component that is created when a CMVC family is configured. All components in a CMVC family are descendants of the root component. Only the root component has no parent component.

SCCS. See *Source Code Control System*

search. The scanning of one or more data elements of a set in a database to find elements that have certain properties.

server. A workstation that performs a service for another workstation. A file that is shared between two or more releases. See also *common file*.

SEI. Software Engineering Institute

shell. Generic name for UNIX command-line interpreter. UNIX shells are also noncompiled procedural programming languages with which end users and system administrators can build utility programs.

Software Engineering Institute. Carnegie-Mellon University 's Software Engineering Institute is a research and development center funded by the United States federal government. Carnegie-Mellon University developed an assessment vehicle that was accepted by the U.S Department of Defense. This assessment allows improvements in the software development processes will help achieve quality, productivity, and cycle-time reduction goals.

Source Code Control System (SCCS). SCCS is a complete system of commands that allows specified users to control and track changes made to an SCCS file. SCCS files allow several versions of the same file to exist simultaneously, which can be helpful when developing a project requiring many versions of large files. The SCCS commands support multibyte character set (MBCS) characters.

state. Tracks, levels, features, and defects move through various states during their life cycles. The state of an object determines the actions that can be performed on it. See also *process* and *subprocess*.

subprocess. CMVC subprocesses govern the state changes for CMVC objects. The design, size, review

(DSR) and verify subprocesses are configured for component processes. The track, approve, fix, level, and test subprocesses are configured for release processes. See also *process* and *state*.

superuser privilege. A user who is granted superuser privilege. Superuser privilege allows a user to perform any action available in the CMVC family.

system administrator. A user who is responsible for all system-related tasks involving the CMVC server, such as, installing, maintaining, and backing up the CMVC server and the relational database being used by the CMVC server.

TCP. Transmission control protocol.

Transmission control protocol. A communication protocol following the U.S. Department of Defense standards for internetworking protocol.

tester. A user responsible for testing the resolution of a defect or the implementation of a feature for a specific level of a release and recording the results on a test record.

track. A CMVC object created to monitor the progress of changes within a release to resolve a specific defect or implement a specific feature.

user. A person with an active user ID and access to one or more CMVC families.

user exit (UE). A user exit allows CMVC to call a user-defined program during the processing of CMVC transactions. User exits provide a means by which users can specify additional actions that should be performed before completing or proceeding with a CMVC action.

UE. User exit

verification record. A status record which must be marked by the originator of a defect or a feature before the defect or feature can move to the closed state. This allows the originator to verify the resolution or implementation of the defect or feature they opened.

version control. The storage of multiple versions of a single file along with information about each version.

view. An alternative and temporary representation of data from one or more tables.

working file. The currently checked-out version of a CMVC file.

List of Abbreviations

3GL	Third Generation Language
4GL	Fourth Generation Language
AIC	AIXwindows Interface Composer
AIX	Advanced Interactive eXecutive
ADSM	ADSTAR Distributed Storage Management
ANSI	American National Standards Institute
APAR	Authorized Program Analysis Report
API	Application Programming Interface
ASCII	American National Standard Code for Information Interchange
BOM	Bill of Material
BOS	Base Operating System
CASE	Computer Aided Software Engineering
CBS	Compound Base System
CM	Configuration Management
CMS	Conversational Monitor System
CMVC	Configuration Management and Version Control
CUA	Common User Access
DB2	DATABASE 2
DCE	Distributed Computing Environment
DEC	Digital Equipment Corporation
DoD	Department of Defense
DOS	Disk Operating System
EBCDIC	Extended Binary Coded Decimal Interchange Code
E/R	Entity/Relationship
FTP	File Transfer Protocol
GB	Gigabyte
GUI	Graphical User Interface
HP	Hewlett-Packard Company
IBM	International Business Machines Corporation
IBM Austin	IBM Development Laboratory at Austin, TX
IBM Toronto	IBM Development Laboratory at Toronto, Ontario, Canada
IBM Tucson	IBM Development Laboratory at Tucson, AZ
IP	Internet Protocol
ISO	International Standards Organization
ITSC	International Technical Support Center
ITSO	International Technical Support Organization
LAN	Local Area Network
LPP	Licensed Program Product
MB	Megabyte
MCI	Microwave Communications Incorporated
MVS	Multiple Virtual Storage
NEDS	Native Environment Development System
NFS	Network File System
NLS	National Language Support
OOA	Object Oriented Analysis
OEM	Original Equipment Manufacturer
OFS	Open Software Foundation
OPATS	Online Problem management And Tracking System
OS	Operating system
OSF	Open Software Foundation
OS/2	Operating System/2
PC	Personal Computer
PMR	Problem Management Record
POSIX	Portable Operating System Specifications
PPS	Production Planning System
PSL	Process Specification Language
PTF	Program Temporary Fix
PVCS	Polytron Version Control System

RDBMS	Relational Data Base Management System
RETAIN	Remote Technical Assistance Information Network
REXX	Restructured Extended Executor Language
RISC	Reduced Instruction Set Computer
SCCS	Software Change Control System
SCM	Software Configuration Management
SEI	Software Engineering Institute
SCS	SEIKO Communications Systems Inc..
SMIT	System Management Interface Tool
SMP	Symmetrical multiprocessor
SNA	Systems Network Architecture
SQL	Structured Query Language
SWSM	Shared Working Space Manager
TBS	Tire Base System
TCP/IP	Transmission Control Protocol/Internet Protocol
TOOAS	Tucson Object Oriented Analysis Simulator
UK	United Kingdom
USA	United States of America
VM	Virtual Machine
VM/AS	Virtual Machine/Application System
VM/CMS	Virtual Machine/Conversational Monitor System
WAN	Wide Area Network
ZAPAR	Zaepfel APAR Tool

Index

P

Q

R

S

T

U

V

W

Z

ITSC Technical Bulletin Evaluation
RED000

International Technical Support Organization
Looking at CMVC from the Customer Perspective
May 1995

Publication No. GG24-4345-00

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction ____

Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) Are you an employee of IBM or its subsidiaries? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

_____

_____

What other topics would you like to see in this Bulletin?

_____

_____

What other Technical Bulletins would you like to see published?

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____ _____
Name Address

_____ _____
Company or Organization

_____ _____
Phone No.

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Department 471, Building 070B
5600 COTTLE ROAD
SAN JOSE CA
USA 95193-0001

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Printed in U.S.A.