Brent O. Hatch (5715)
HATCH, JAMES & DODGE
10 West Broadway, Suite 400
Salt Lake City, Utah 84101
Telephone: (801) 363-6363
Facsimile: (801) 363-6666

Stephen N. Zack (admitted pro hac vice)
Mark J. Heise (admitted pro hac vice)
BOIES, SCHILLER & FLEXNER LLP
Bank of America Tower – Suite 2800
100 Southeast Second Street
Miami, Florida 33131
Telephone: (305) 539-8400
Facsimile: (305) 539-1307

Robert Silver (admitted pro hac vice)
BOIES, SCHILLER & FLEXNER LLP
333 Main Street
Armonk, New York 10504
Telephone: (914) 749-8200
Facsimile: (914) 749-8300

Frederick S. Frei (admitted pro hac vice)
Aldo Noto (admitted pro hac vice)
John K. Harrop (admitted pro hac vice)
ANDREWS KURTH LLP
1701 Pennsylvania Ave. NW, Suite 300
Washington, DC 20006
Telephone: (202) 662-2700
Facsimile: (202) 662-2739

Attorneys for Plaintiff The SCO Group, Inc.

FILED IN UNITED STATES DISTRICT COURT, DISTRICT OF UTAH
JUL 12 2004
MARKUS B. ZIMMER, CLERK
BY DEPUTY CLERK

IN THE UNITED STATES DISTRICT COURT

FOR THE DISTRICT OF UTAH

THE SCO GROUP, INC. Plaintiff/Counterclaim Defendant vs. INTERNATIONAL BUSINESS MACHINES CORPORATION Defendant/Counterclaim Plaintiff	**UNDER SEAL** **DECLARATION OF CHRISTOPHER SONTAG IN SUPPORT OF SCO'S REPLY MEMORANDUM REGARDING DISCOVERY** **Case No. 2:03-CV-0294 DAK** Honorable Dale A. Kimball Magistrate Judge Brooke C. Wells

unsealed

203

1. My name is Chris Sontag and I am Senior Vice President and General Manager of The SCO Group, Inc. My office is located in Lindon, Utah. Unless otherwise noted or evident from their context, this Declaration is based on my personal knowledge and information available to me from reliable sources. To the best of my knowledge, information and belief, the facts set forth herein are true and correct.

2. This Declaration responds to and rebuts the position asserted in IBM's Response to SCO's Memorandum Regarding Discovery, and in the Declaration of Joan Thomas ("Thomas Decl.") [Exh. A to IBM Response ("IBM. Resp.")].

3. IBM claims that producing the materials SCO requests will take "months". *Thomas Decl.* ¶ 3. This is grossly exaggerated. Producing the materials SCO requests, almost all of which are available on IBM's computerized Configuration Management Version Control (CMVC) system, should not take more than a few weeks.

A. IBM's Configuration Management Version Control (CMVC)

4. SCO needs IBM to produce all revision control system information (including documents, data, logs, files, and so forth) for AIX and Dynix/ptx from 1984 to the present, and log information for all interim and released versions of AIX and Dynix/ptx from 1984 to the present, in a usable, searchable format. Specifically, SCO requires IBM to produce:

A. Source Code Control System's (SCCS) data files related to AIX, in the same format and organization as stored in CMVC and

B. the SCCS directory hierarchy used by CMVC,

all on standard format (ISO-9660 with RockRidge extensions) DVDs; as well as

C. Any other AIX- and Dynix/ptx-associated information stored on CMVC and/or RCS.

5. According to IBM, the CMVC system "provides configuration management, version control, change control, and problem tracking in a distributed environment to facilitate project-wide coordination of development activities across all phases of the product development life cycle." *CMVC Introduction* [*Initial Mem.*, Exh. 7]. In short, CMVC maintains all versions of a particular program, and tracks changes to and problems with that program entered by multiple users.

6. CMVC can be accessed at remote locations, and includes tracking information about source files, defects, and features. *Overview of AIX Source Control* (1710013143)[Exh. 1].

7. CMVC also records author information, so that every time a programmer enters information onto CMVC, the programmer's name is recorded, as well as the date of the entry and what action was taken. IBM's Redbook, *Did You Say CMVC?*, Sept. 1994, at 9 [Exh. 2]; Email from Adrian Mitu of IBM Canada, Aug. 19, 1993 (located at <http://www.cs.queensu.ca/Software-Engineering/blurb/cmvc>) [Exh. 3].

8. According to at least one IBM employee, several functions of CMVC are designed for fast, simple access to revision history of software programs. For example, the "VC" (Version Control) function of CMVC "eliminates confusion when looking for a particular source code file belonging to a particular product". Email from A. Mitu [Exh. 3].

9. Another IBM employee explains the "advantage of CMVC is its versatile development process that can be configured for small, medium and large projects. The process

can also be configured to meet the needs of each stage of development. . . ." *A Versatile Development Process for Small to Large Projects Using IBM CMVC*, Seong R. Yu [Exh. 4].

10. CMVC information for AIX is contained on a single CMVC server at IBM. *Thomas Decl.* ¶ 7.

11. IBM exaggerates the burden it faces in producing the CMVC information SCO requests. CMVC is designed so that multiple engineers may easily access the system, log in and out with changes and modifications, and continue to develop AIX without delay or disruption. *See CMVC Introduction* [*Initial Mem.*, Exh. 7]. If CMVC could work only in the extremely slow way IBM alleges, IBM's daily business operations would be hindered.

12. IBM uses the flexibility and sophistication of CMVC in its marketing efforts, claiming that IBM uses automated scripts to incorporate fixes and changes daily. *IBM-Siebel Marketing* [Exh. 5] at 23.

13. IBM has produced a limited set of the prior versions and releases (and copies only of currently supported versions: 4.3, 5.1, and 5.2), and few if any maintenance modifications.

14. By selecting only certain versions to produce, IBM created more work for itself as IBM had to search its CMVC system for the correct code that constituted the produced versions and releases. IBM's decision to extract only selected releases of AIX and Dynix/ptx for production was more burdensome for IBM to carry out than simply giving SCO access to all the CMVC information.

15. There are often two methods of solving a particular problem in computer science: "top down" and "bottom up". In Thomas' description of IBM's burden, IBM is proposing to use a bottom-up approach, which is overly laborious, and an unnecessary expenditure of time and

resources. IBM's bottom-up approach requires IBM to look at all parts of its CMVC database to find directories that SCO wants.

16. Because CMVC has a database component, IBM's approach requires it to query the database to find the required directories by specific file name. The problem with this approach is that the database query may find too many directories.

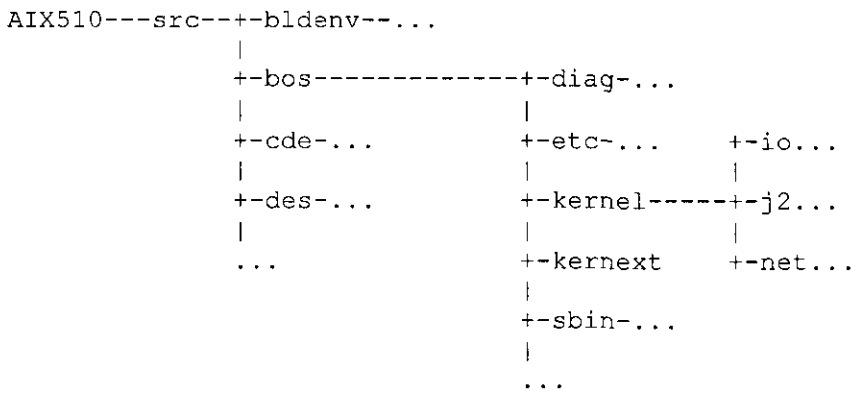
17. As an analogy to the burdensome task IBM assigns itself, when searching on Google, a large number of search matches may be returned, some of which *do not satisfy* the constraints the searcher wants, but *do satisfy* the exact wording of the request. Thus, the search results must often be refined to match the searcher's constraints more precisely; otherwise, many unsatisfactory matches must be examined needlessly. The same is true of IBM's proposed solution to the SCO's requested CMVC discovery.

18. However, continuing the Google analogy, if the searcher instead knows the website he is searching for, he would simply go to that web page directly, instead of using Google. Only if the website option is not available are the extra burdens of searching required. Here, the extra burdens of searching are not required because there is "website option"; that is, CMVC overlays IBM's SCCS hierarchy, so IBM can collect all AIX information based on that hierarchy using directory names. *Thomas Decl.* ¶¶ 9-10. No searching is required, simply a look-up of AIX files by the name "AIX."

19. This "top-down" approach will not only work, but will be sufficient and is the most time- and labor-efficient means for IBM to collect the CMVC information SCO requires.

20. For example, consider the following file tree (which is a part of the AIX tree for AIX version 5.1), produced by IBM:

FIGURE 1



21. There are 38 subdirectories under the "src" directory in AIX510, four of which are shown in Figure 1 as examples (bldenv, bos, cde, and des). One of these is "bos," probably an abbreviation for "basic operating system". There are 10 further subdirectories under "bos," 5 of which are shown in Figure 1 as examples (diag, etc, kernel, kernext, and sbin).

22. The file tree in Figure 1 shows the layout of SCCS for all versions of AIX, meaning that AIX directories are easily recognizable.

23. The CMVC system overlays SCCS hierarchy. Thus, CMVC, organized by IBM's SCCS hierarchical method, looks similar to Figure 1. Therefore, any directory (or "tree limb") associated with "AIX" (here, it is called AIX510) will fall within SCO's requests, and should be produced. This method of seeking and collecting directories responsive to SCO's requests is a top-down approach.

24. The CMVC system is analogous to the Lexis/Nexis or Westlaw systems. Each is a database system containing data to be accessed remotely by multiple users. Searches are conducted in each system to locate particular data maintained in each database. Each uses tools

for user interfacing, allowing searches of the databases. Each court decision recalled from Lexis/Nexis or Westlaw is like a computer file stored as text in CMVC. For example, searching for AIX on CMVC might be compared with searching the Utah State Reporter System. Within the AIX directory/subdirectory tree, searching for the directory "src" might be compared with searching a particular Utah district. A subdirectory may be searched which would be analogous to searching a Judge's name.

25. IBM's internal CMVC documents prove that SCO's top-down approach will be significantly less burdensome than the approach IBM assumes. IBM's *Overview of AIX Source Control* (1710013143) [Exh. 1] reveals that CMVC material is generally maintained in a hierarchical structure. IBM's SCCS, based on a file tree structure, can find all directories associated with AIX stored in SCCS format on CMVC.

26. IBM could write a "script" (as IBM defines it, a small computer program) (*Thomas Decl.* ¶ 9) to seek out and collect the relevant CMVC information. For SCCS hierarchical data, writing this "script" and running it should take no more than one day (not weeks or months), and for competent IBM engineers familiar with CMVC, probably less.

27. In the alternative, IBM can use standard archival tools to determine the locations of the SCCS directory hierarchies related to AIX, and make copies. These tools basically say "I want everything under these directories" and then copy all of the files in those directories. Thus, this procedure is neither burdensome nor time-consuming.

28. If IBM follows this top-down approach to collecting the information SCO has requested, the burdens IBM describes in its Response and the accompanying Declaration of Joan Thomas – as to time and effort – will be significantly lessened.

29. IBM asserts that it will need to confirm that the directories extracted from CMVC are actually part of the AIX operating system. *Thomas Decl.* ¶ 9. Especially for SCCS hierarchical data, this review is entirely unnecessary, and serves only to falsely inflate IBM's estimate of the time and burden required to produce the materials SCO requests. First, these files will be produced under a protective order.

30. More importantly, if the files are labeled in CMVC as being part of AIX – or otherwise identified as being required to do any of the AIX releases – then the directories are relevant and responsive to SCO's requests, and should be produced.

31. IBM also provides a misleading time estimate in stating that to copy AIX information from data tapes onto DVDs or CDs would take "many additional weeks", after the "many weeks" already taken to collect the information. *Thomas Decl.* ¶ 11. This is grossly exaggerated.

32. Ms. Thomas admits in her Declaration that all files on CMVC containing AIX operating system source code constitute only 40 gigabytes of data. *Thomas Decl.* ¶ 10. This amount of data will easily fit onto a typical laptop computer (running UNIX) or a typical portable hard drive, both of which have at least 40-60 gigabytes of disk space, or onto 10 DVDs.

33. It takes less than 1 hour to burn a single DVD, and transferring AIX data from CMVC onto 10 DVDs should take approximately one day. Therefore, the most efficient alternative for both SCO and IBM is for IBM to copy the AIX directories onto ten standard format (ISO-9660 with RockRidge extensions) DVDs, as this standard format can easily made by IBM and easily used by SCO, and does not take a great deal of time to complete.

34. In sum, it should not take IBM more than two days to extract and copy the relevant SCCS hierarchical data onto 10 DVDs.

B. Format of Tapes Produced by IBM

35. IBM claims that the "high-capacity storage tapes" it initially produced to SCO "are standard throughout the computer industry." *IBM Resp.* at 10; *Thomas Decl.* ¶ 11. This is not entirely accurate. While many sites still use the tapes used by IBM, ISO-9660 DVDs or CD-ROMs are the current standard interchange format, and IBM's initial production was not in this format. Since IBM's tapes and ISO-9660 DVDs each hold roughly the same amount of information, and since DVD burners are far less expensive than tape drives, it should be no hardship for IBM to produce the data in DVD form.

C. Sequent's Revision Control System (RCS)

36. Sequent used a system similar to IBM's CMVC and SCCS called Revision Control System (RCS) to track the revision history of Dynix.

37. IBM does not state where it maintains revision history information for Dynix/ptx, or whether IBM still uses RCS. Thomas' Declaration states only that Dynix information is not maintained on CMVC. *Thomas Decl.* ¶ 5.

38. RCS is an open-source program, and easily available to IBM.

39. IBM has not stated that the Dynix/ptx source code RCS database is shared with any other projects, so IBM should be able to simply copy the entire Dynix/ptx RCS database without review or extraction, and without taking any of the steps it claims are required for the

AIX CMVC source code database. Even if RCS is not currently maintained, unless IBM has intentionally deleted the RCS database, the database can be provided without burden. Therefore, IBM would not bear any burden in producing Dynix/ptx information, if that database still exists or has been archived.

D. Programmer's Notes and Design Documents

40. Programmer's notes and design documents are often stored in the CMVC or SCCS change logs. Ms. Thomas confirmed that IBM stores programmer's notes and design documents on CMVC. *Thomas Decl.* ¶ 12. Therefore, these programmer's notes and design documents can produced as part of IBM's production of CMVC data.

41. As explained in my 56(f) Declaration (¶ 53), programmer's notes contain the thought processes of individual programmers as they write and revise code sequences. For example, programming notes might list changes made to code, and might list additional changes to consider. As such, programming notes provide detailed rationale for code changes and an indication of how the code may change in the future. Programming notes are often formatted as "readme" files that are saved in the same directory as the corresponding source code files.

42. As explained in my 56(f) Declaration (¶ 52), design documents are often prepared by the group that ultimately authors the changes to the code sequences, and explain the initial code concepts, and how such code will be developed and written. As such, design documents provide an invaluable bridge between existing code sequences, such as in UNIX, and derivative works, such as in AIX and Dynix.

43. Therefore, programmer's notes and design documents might contain admissions as to reliance on structure, sequence and organization that could connect UNIX System V to modifications in AIX and Dynix/ptx, and thus point SCO to hot spots of copying in Linux.

44. Programmer's notes and design documents may significantly shortcut the other procedures of painstaking manual line-by-line code comparisons which SCO has used and continues to use to find evidence relevant to its claims and defenses, which are extremely time- and labor-intensive procedures.

45. SCO has used tools that are capable of taking minor changes in lines of code into consideration in finding literal copying. Ultimately, however, these tools can provide only an indication of where an experienced programmer should look in order to perform an intelligent manual comparison. Consequently, extensive manual work is still required.

46. With production of the materials SCO's requests in its Initial Memorandum, SCO's need to use these onerous tasks will be significantly lessened.

I declare under penalty of perjury that foregoing is true and correct.

Executed: July 12, 2004

Christopher Sontag

CERTIFICATE OF SERVICE

I hereby certify that a true and correct copy of Chris Sontag's Affidavit in Support of Reply Discovery Memorandum was served on Defendant International Business Machines Corporation on this 12th day of July, 2004, by placing it in U.S. Mail, first class postage prepaid, to:

Alan L. Sullivan, Esq.
Todd Shaughnessy, Esq.
tshaughnessy@swlaw.com
Snell & Wilmer L.L.P.
15 West South Temple, Ste. 1200
Gateway Tower West
Salt Lake City, Utah 84101-1004

Evan R. Chesler, Esq.
echesler@cravath.com
Cravath, Swaine & Moore LLP
Worldwide Plaza
825 Eighth Avenue
New York, NY 10019

Donald J. Rosenberg, Esq.
1133 Westchester Avenue
White Plains, New York 10604

EXHIBIT 1

The following information describes the framework that IBM currently utilizes to manage the AIX Operating System and related product source code. This framework facilitates distributed development involving remote IBM sites as well as external companies while maintaining a common AIX product source tree.

This information is being supplied for general working knowledge, is non-binding and in no way represents a commitment from IBM to the reader. Any arrangements regarding working relationships, source access, licensing, joint development and derivative works must be subject to the provisions of a definitive agreement executed by authorized representatives of IBM and applicable parties.

Overview of AIX Source Control

AIX development utilizes two major systems to develop and control AIX source. These are the CMVC product (Configuration Management and Version Control) which is used to manage changes to AIX source code and the ADE (AIX Development Environment) which is used to support development and build of the AIX source. The CMVC system is a client-server system where the server system maintains the database of source files, their deltas along with their change history and associated defects and features. The CMVC client is supported on many different platforms and can be used at remote locations to obtain, create and modify source as well as manage defects and features. A defect is a CMVC record opened against a particular product or product component for the purposes of reporting a bug in the product that needs a fix. A CMVC feature is a record opened against a particular product for the purposes of enhancing the product.

CMVC provides a very flexible report management system to provide information concerning source file, defect (bug) and feature (design change / enhancement) information tracking information.

The official AIX source is stored in CMVC managed databases at IBM Austin. Sources are organized within CMVC by product and within each product by components. Components are arranged in a hierarchical structure with a single top component called root. The component structure reflects organizational requirements for the development of AIX and related products. It is different and separate from the actual product binary structure, and from the source file tree organization used for file location and building. At any point in time, ownership and control of a particular part of the product is determined at the component level. All files in the product have a designated component, and change authority is managed at the component level. The component owners can be local to IBM Austin, remote sites within IBM, or external to IBM. Each component owner has control over whom within their area may access the component source and at what privilege level. CMVC enforces access control at the component level.

Many releases of AIX exist in CMVC as delta sources. Some are dedicated to servicing of current software versions available for customers and others are for software under development. There is only one release per product in CMVC at any point in time that is in development phase. When the stabilization of this development version is required prior to final product testing, a new CMVC release tree is created for *future development.*

During product development, CMVC is typically used with Source Control and Change Control binding activated. Binding Change Control links all file changes to a reported problem (CMVC defect) or proposed design change (CMVC feature). When a developer wishes to make a change to a source file, the source is checked out of CMVC (locked to prevent others from modifying the file). When a modification has been coded, inspected and unit tested the sources are checked into the CMVC repository in Austin creating a new delta of the file tracked by a version number (SCCS ID). At the file check-in time, developers must relate the change to a defect or feature number.

CMVC provides a mechanism to track which modification requests are implemented for one release. This mechanism defines "tracks" and "levels".

Each defect or feature that will result in a source file creation or modification will have a track associated with it. The defect or feature owner can create the track, and specify all components that are expected to make source changes relating to the track. Developers that are making changes to source files in support of

the defect or feature, check-in the updated source file by specifying the track which is associated with a particular product release.

A selected set of fix or feature tracks for a specific product and release are grouped in a level in order to do a build once all components specified in the track have been marked "complete" by the component developers.

Overview of ADE

The AIX Development Environment is designed to allow simultaneous development on multiple releases of one or more product sources. The ADE is an enhanced version of the ODE (OSF Development Environment) tools. It is based on the standard Unix make command and Makefile but has been enhanced to support two main types of workspaces:

- Sandbox: This is a private working area which a developer owns and controls. Sandboxes contain both source and object files that have been modified by the developer, but have yet to be included in a product build. Sandboxes can be shared between a team of developers.
- Backing Tree: This is a common source and object-backing tree for all developers. It contains a full set of product files, objects and executables. The developer selects which backing tree to build their sandbox against. A sandbox is backed to a backing tree or to another sandbox. During a private or production build, when an action requires a file that is not in the sandbox, ADE searches the backing tree of the sandbox for the file.

A sandbox usually contains only the files the sandbox user wants or has modified. The backing tree is a chain of sandboxes with the root of the chain being the backing build. The backing build is usually the last published production build. All development and build tools comprising ADE are included in the production build (thus backing build), so that all developers are utilizing the same level of tools.

Overview of Build Management

There are typically three types of builds performed. Production builds which are performed by the Austin based BAI (Build and Integration) group, and area builds that are typically down by the local development groups and sandbox builds that are done by one or more developers. Developers do their sandbox builds to verify the correctness of their changes before they check the changes back into CMVC. Once the changes have been checked into CMVC, the local area build team collects the lists of tracks from the area (and any dependent tracks from other groups) and puts those tracks into a level. That level is then extracted from CMVC which pulls all of the changed files into a area build sandbox, and the sandbox is built. A sniff test is performed by the local area builder to ensure the basic function is maintained. If all of these operations are successful, the area build is deemed successful and the area builder provides the level (list of all tracks) to Austin BAI. BAI then collects all successful levels from the area builders (typically on a regularly scheduled build interval), extracts those levels from CMVC into a BAI sandbox and performs the build. Once the build is successful a build verification test is run to ensure the product of the build is basically functional. If not, the build group opens severity 1 defects against the components introducing problems. Once the test is successful, BAI commits the level in CMVC which moves all tracks in the build to committed state, which then results in all track owners receiving test records to verify that their changes are correct. Once committed, tracks cannot be modified, new defects must be opened to make corrections.

The complete tree containing sources, binaries, tools and the bootable system is deemed a Production build and published to IBM sites using DFS to be used as their latest backing tree for continued development.

For external sites (not having DFS access) the extraction tool used to extract build source from CMVC has the capability to use the origin information associated with each file to facilitate the external sites compliance with licensing restrictions. In cases where this makes the source not build-able, the tool injects appropriate stub files to ensure the build is successful. These source files are then packed into a format acceptable to transmit them to the remote site, where they are extracted and built locally to provide the local backing build.

EXHIBIT 2

Did You Say CMVC?

Document Number GG24-4178-00

September 1994

International Technical Support Organization
San Jose Center

1.4 Automated Support for SCM and Change Management

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

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

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

1.4.1 Configuration Management Version Control

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

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

1.4.1.1 Configuration Management

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

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

1.4.1.2 Version Control

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

1.4.1.3 File Change Control and History

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

1.4.1.4 Integrated Problem Tracking

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

1.4.1.5 Release Management

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

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

1.4.1.6 Access Control

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

1.4.1.7 Automatic Notification

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

1.4.1.8 Customization

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

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

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

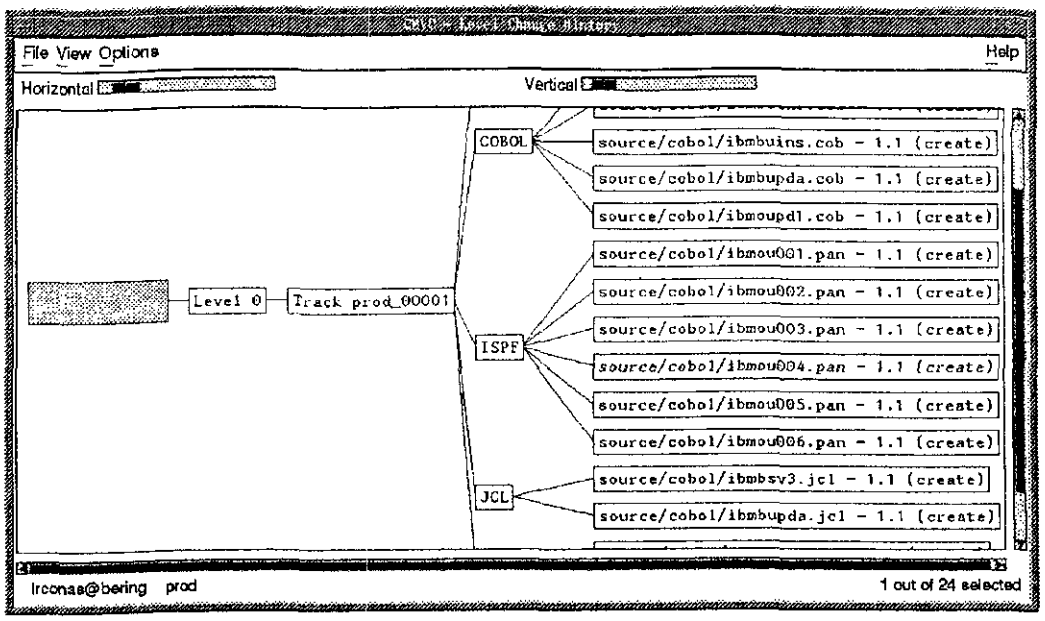


Figure 24. Level Change History

The application source files, managed by SCCS, are stored on the file system associated with the AIX family login name, *prod*. The *prod* family audit log file (/production/audit/log) traces all CMVC transactions that were performed.

Chapter 4. Planning for CMVC

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

4.1 Why Plan?

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

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

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

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

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

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

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

4.2 Pre-Installation Planning for CMVC

Before you install CMVC, you should:

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

4.2.1 Planning Network License Requirements

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

4.2.1.1 What to Consider in Planning Network License Requirements

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

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

off as parameters to a CMVC command that is issued from the Development Manager CMVC pull-down menu, if that command involves the same component and release tuple associated with that context mapping. The purpose of this arrangement is so that end users can work in directory hierarchies that mimic the path names associated with files under CMVC control, and have the Development Manager prefill CMVC window fields (which require path and file names) with the correct path names. It is very important to master this concept and use context mappings, because CMVC can get confused if you add these path names by hand where they should be prefilled. Context mapping is described in more detail than we provide here in "The CM Class Messages" of "Using the Message-Integrated CMVC GUI" of *IBM CMVC User's Reference*

It is not recommended that users edit the *.cmvcrc* file. If you want to clear a single mapping default, clear the component and release fields from the Context Mapping dialog box when it is being prompted. However, if you want to delete some mappings, leaving others, or simply verify the mappings that you have defined, edit this file.

E.3 Implications of Host Scoping for CMVC

CMVC is not designed to be "network aware" in that it does not *know* it can concatenate the path name `/nfs/hostname` (using the data context's host name, parameter passed to it by Execution Manager) to the data context's path name parameter, to find a file on a remote host. The CMVC client merely compares the host parameter passed to it against, the host on which it is executing, and if they do not match, CMVC issues an error message.

We infer that CMVC designers made this design decision, because CMVC is a host scoped tool. But we do not think that being host scoped necessarily implies a tool is not network aware. Other tools, such as the Build Tool, which is network scoped, and the Development Manager, which is directory scoped, are network aware, and can locate remote data, assuming the proper file systems are exported and mounted. Host scoping simply means that only one instance of the tool can be invoked automatically by the Execution Manager per host.

If you have multiple CMVC clients installed on your network, and want to have CMVC access data on them, you must configure the CM class tool through your personal `.softinit` file or the system-wide `/usr/softbench/config/softinit` file to execute on whatever host is identified by the data context parameter passed to it. The value %Host% goes in the execution host field. When this is the case, then the Execution Manager starts up an instance of the CM tool on the remote host, if it recognizes a remote host in the data context being passed to CMVC. The CMVC client, executing remotely, binds to the local X server and display its results where they belong. CMVC starts up multiple instances of the CM tool under these circumstances, if different data hosts are indicated by subsequent user requests for CMVC actions.

If you have CMVC client only installed on the host where Execution Manager is running, then you must configure the CM class tool to execute only on the local host. To do this, place the value %Local% in the execution host field of the CM tool entry in the `.softinit` file or cause your system-wide `softinit` file to be updated this way. This forces Execution Manager to assume that only one instance of the CM tool should be started up, and to always start it on the same host it on which Execution Manager is executing.

When the execution host is set to %Local%, Execution Manager ignores the host portion of the data context. However The CMVC client does pay attention to the host portion. If you have asked it to deal with data on a remote host, it will balk. This can happen if you set the Development Manager to a path beginning with /nfs/hostname/....

If you really only have the CMVC client installed on the host on which your users execute Execution Manager, and you want to access remote files made available to your CMVC client machine through NFS, do not access them by setting the Development Manager context to a path name that begins with /nfs/hostname, or by explicitly naming remote data host. Development Manager is network aware, and infers from this path name that the data context is in fact on a remote host, and sets up the data context parameters accordingly. CMVC will then refuse your request (if CMVC were network aware, it could look to see if the data context were available at the default NFS path and access it there, but this is not the case as of CMVC Version 2, Release 1). Instead you should set up a link (whose path name does not include /nfs/hostname) to a directory mounted in this default NFS path, and set your Development Manager data context to this directory. Under these circumstances, the Development Manager is unaware that the data is remote and passes data context to CMVC as a local path name. Execution Manager then starts up a local CMVC instance.

The default execution host setting for CMVC is %Local%. This is briefly mentioned in "Broadcast Message Server" of *IBM CMVC User's Reference*. CMVC installation procedures appear to place a file, named softcmvcinit, in the /usr/softbench/config/softinitsrc/class-defaults file, but may not actually execute the /usr/softbench/etc/merge-init file, which merges all tool vendor contributions into /usr/softbench/config/softinit file, which is the system-wide configuration file. This is the recommended approach to tailoring your system-wide softinit settings. See "Customizing Tool Initiation" in *Installing IBM AIX SDE WorkBench/6000 and IBM AIX SDE Integrator/6000* for details on this file. You may be confused if you are not familiar with this process. Follow the directions given in "Installing the CMVC Clients" of *IBM CMVC Client Installation and Configuration*, which explicitly advise the system administrator to edit the /usr/softbench/config/softinit file directly.

E.4 CM Tool Messages

The SDE WorkBench/6000 integrated CMVC client may be activated through BMS messages by various tools that request files be checked in or out, releases be created or extracted, and context mapping be changed. It responds to the general tool messages sent by Execution Manager to start, stop. This message traffic can and should be observed if you, or your users, experience problems, or situations they do not understand, resulting from tool interaction. Refer "The Message Interface" in *Configuration Management Version Control User's Reference, Version 2 Release 1* if you need to determine which messages CMVC will respond to and what the parameters to the messages represents. Many problems may be caused by a misunderstanding on the part of the new SDE WorkBench/6000 user about the data and execution contexts involved with the tools being exercise in a distributed environment. Analysis of the message traffic readily identifies what data is being passed to what tool, and what precise action is being requested.

EXHIBIT 3

Date: Thu, 19 Aug 93 16:52:36 EDT
From: mitu@vnet.IBM.COM
To: dalamb@qucis.queensu.ca
Subject: Addition to software-eng FAQ

From: Adrian Mitu
Internet: mitu@vnet.ibm.com
VNET: TOROLAB6(MITU)
Subject: Addition to software-eng FAQ

David,

Can you please add the following product description to the CM products section of your FAQ?

Thanks!
Adrian.

Configuration Management Version Control (CMVC)
==========

Product Description
==========

CMVC tightly integrates the following functions:

1) Configuration Management (CM)
Software Configuration Management (SCM) is the process of identifying, tracking,
and controlling changes to software configuration items (SCI). SCI's in CMVC terminology are called components. A configuration consists of a family of components. CMVC components can represent any objects that need to be managed: software modules, documentation sets, design requirements, object code, etc. Within the family, the components are arranged in a tree-like hierarchical structure that reflects the structure of your organization and the structure of the projects you are working on.

2) Version Control (VC)
Version Control manages all the versions of all the SCI's. Any version of any SCI is available at any time. The mechanism used is forward-delta versioning, that enables version branching with a minimum of storage wasted. Both ASCII and binary files can be stored under CMVC's Version Control mechanism. All of the SCI are available to users from a CMVC server, although their physical location can be distributed. This eliminates confusion when looking for a particular source code file belonging to a particular product, for example.

3) Problem Tracking (PT)
Problem Tracking works through two mechanisms: Defects and Features. Defects will track the lifecycle of a problem with any of the SCI's from identification all the way through to final resolution. CMVC introduces and enforces a methodology for resolving Defects and implementing Features. An established software development process is essential for improving the quality of the product. Moving your organization from a low level of process maturity to a higher one is dependent on everybody in the organization knowing and following a repeatable process. CMVC makes it easy to follow and learn the process by notifying the users who need to take some action that they must provide some input, or take certain actions.

4) Change Control (CC)

Change Control provides an audit Track that identifies which files have gone through what changes, when, who modified them, and why. A Defect or Feature may result in the modification of hundreds of files across more than one product, including documentation and testcases. The Change Control process, through the mechanism of Tracks, will produce and keep records of each file modified in each separate product as related to the particular defect or feature. When you are ready to include the changes for the defect in the release and build the release, you only need to specify the Track number (same as the Defect number) and all the changes will be committed to the database and included in the release. Product baselines (called Levels in CMVC's terminology) are created by specifying which fixed defects and implemented features are to be included. All the changes associated with a particular defect or feature are therefore treated as a unitary set.

5) Access Control (AC)

Access control is done at the CMVC component level. Access authority is inherited downward from parent components to children components. There is a fine grain of access permissions that can be configured in CMVC, going far beyond typical read/write permissions. Access is also granted automatically by CMVC to authorized users who must take certain actions. For example, a developer cannot normally change source code files, but when he is responsible for fixing a defect, CMVC automatically gives him permission to modify the files belonging to that particular release, until the developer indicates to CMVC that he has finished fixing the defect.

6) Automatic Notification

All the users who must take some action are notified by CMVC automatically. Also all the users who wish to be notified (for interest) of changes in CMVC, will be notified by CMVC. The level of interest is configurable by component and by action. The notification is done via electronic mail, so users do not have to 'log in' to CMVC to find out about that they must take an action or about the changes they are interested in.

7) Release Management

A CMVC Release groups together a set of files belonging to one or more components into a logical unit. Files and components can be shared by more than one release. This feature encourages software reusability between different
projects. A maximum of commonality is maintained even if the versioning of a particular file is split by the projects that are using it. CMVC Releases allow you to extract an older version (e.g. a previous release) of your source code and work on that (as opposed to on the most recent) for fixing a problem or deriving a new strand of your product. Baselines (called Levels in CMVC) can be defined as snapshots of a Release. Also, levels can be classified as development levels, productions levels, etc. . You can extract all the files in any level at any time, and you can also extract the differences between two consecutive levels.

Other features of CMVC include:

Client-Server architecture

All the files and the data reside on one server (although it is possible to distribute the stored files over a LAN), eliminating the search for 'the right copy of file X' and minimizing the backup effort. The users access the server through the CMVC clients.

Multiple Hardware Platforms Supported

The CMVC supports: RISC/6000 (AIX 3.2.x), Sun SPARCstations (Sun OS4.1.x) HP Apollo 9000 series 400, 700 and 800 (HP-UX 9.x).

Ports to other platforms are under way for both the
server and the client. All the CMVC clients and servers can co-exist in
a totally heterogeneous environment.

Integration through SDE Workbench or Softbench

CMVC uses BMS to provide integration with any tool in the SDE Workbench or
Softbench environment.

Graphical Interface

CMVC's Motif graphical interface makes using CMVC a snap. It provides intuitive
panels and graphical views of the configuration stored and of the project
status. The graphical interface looks identical across all the platforms
supported. A command-line interface is also provided.

****
*CMVC is an IBM product. Contact your (any) IBM rep for pricing and product *
*info. If you can't get hold of an IBM rep, you can send me e-mail at *
*mitu@vnet.ibm.com, and I'll try to locate one. You can also send me e-mail *
*if you have technical questions. *
****

==========

Adrian Mitu
tel:(416)-448-6177
fax:(416)-448-4414

IBM Canada Ltd.
1150 Eglinton Ave East, Toronto,
Ontario, M3C 1H7, CANADA

EXHIBIT 4

A Versatile Development Process for Small to Large Projects Using IBM CMVC

Prepared by: Seong R. Yu (Steve)
IBM Santa Teresa Laboratory
555 Baily Avenue, San Jose, CA 95141

Abstract

This paper discusses how IBM* Configuration Management Version Control (CMVC) can be applied to an Incremental Development Model, and it shows how CMVC can be configured to match different project sizes and different stages in the software development life cycle. Applying too much process during a stage in the software development lifecycle can cause unnecessary work, and too little process can cause lose of control. Applying the correct amount of process will help to achieve target quality and productivity goals.

Introduction

At the IBM Santa Teresa Laboratory (STL), the size of software development projects can vary from a small team of few developers to a large team in excess of two-hundred developers. The projects have different target cycle times, support multiple platforms, use different combinations of programming languages, and can all exist at different stages in the product life cycle.

Configuration Management Version Control (CMVC), an IBM licensed program[1], has been selected as a strategic tool to STL's needs. Table 1 shows the different tools deployed at STL prior to establishing site wide CMVC support.

CMVC was chosen for the following functions [4]:

- Configuration Management
- Version Control
- Change Control
- Problem Tracking
- Project-wide Coordination
- Distributed Development Environment

The advantage of CMVC is its versatile development process that can be configured for small, medium, and large projects. The process can also be configured to meet the needs of each stage of development: Prototype, development, test, preship, maintenance, and emergency fix. New processes can also be created using a combination of processes shipped with CMVC.

Ready accessibility, ease of use, low support and maintenance cost, scalability, configurable process, and high availability are

[1] IBM Configuration Management Version Control/6000 (Program 5765-207)

the support characteristics of the STL Site CMVC. Projects using CMVC will benefit from improved quality [1] and productivity.

CMVC satisfies six elements of ISO9001 [2]:

- Design Control
- Document Control
- Product Identification and Traceability
- Inspection and Test Status
- Control of Nonconforming Product
- Internal Quality Audits

The large product development projects, such as IMS and DB2* for the MVS platform, have a longer history at STL as compared to the smaller workstation development projects for the OS/2* platform and more recently for the AIX* platform.

Many of the earlier processes in use by STL were centered on large mainframe projects such as the business and product planning process, development process, and distribution and service process. The first workstation projects struggled to define the right development process and to find supporting tools. Adapting the same development process used for the large mainframe projects would have been too burdensome.

TABLE 1

TOOLS DEPLOYED AT SANTA TERESA LAB PRIOR TO CMVC

Activity	Platform	Library System	Problem Tracking	Design Change Tracking	Build	Service
Prototyping	All	none	none	none	none	none
Code Development	MVS	LCS[2]	FPTM[3]	DCR[4]	CLEAR[5]	SPA[6]
	VM	LCS	FPTM	DCR	CLEAR	SPA
	OS/2	PVCS[7]	none	none	MAKE	none
	AIX	SCCS[8]	none	none	MAKE	none
Information Development	All	IDPS[9]	none	none	IDPS	IDPS

The scope of this paper is limited to the discussion of the development process and CMVC. The components of the development process in this paper refer to source-code change control, problem management, and design-change management. A simple process is better suited for a small project and for all projects at initial stages. A more elaborate and rigorous process is necessary to coordinate a large number of developers working on a large product in later stages of development and in maintenance stages. CMVC is an excellent tool that can adapt to these environments.

2 LCS - Library Control System (IBM IUO)

3 FPTM - Problem reporting and tracking system

4 DCR - Design Change Request

5 CLEAR - Control Library Environment And Resource system (IBM)

6 SPA - Service Process Architecture (IBM)

7 PVCS** - Polytron Version Control System (trademarked product of INTERSOLV Inc., an IBM Business Partner)

8 SCCS - Source Code Control System (shipped as part of AIX)

9 IDPS - Information Development Publishing System (IBM IUO)

Applying CMVC to Incremental Development Model

An example of an incremental development model, also called an evolutionary model [3], is shown in Figure 1. In this model, a programming project begins with an initial planning phase followed by a series of increments. During the planning phase, decisions are made concerning the number of increments, target dates and schedule, resource assignments, and programming objectives for each increment.

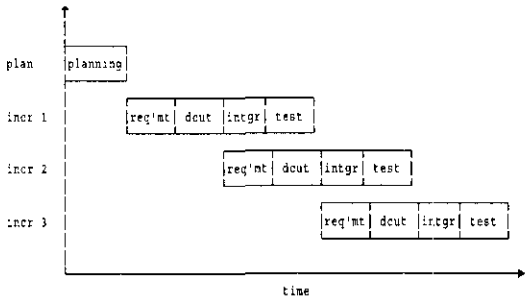


Fig. 1. Incremental Development Model

The programming objectives are derived from requirements such as request to add new functions and features, enhance an existing function or feature, provide solution to previously reported problems, or change design. A good increment plan will yield a running system at the end of each increment.

In the model shown in Figure 1, there are three increments. Each increment consists of four phases: 1) requirements analysis phase, 2) design, code, unit test (dcut) phase, 3) build and integration phase, 4) test phase. At the start of each increment, further refinements and adjustments to the schedule and resource assignments can occur. During the requirements analysis phase of each increment, the requirements and programming objectives are further broken down into more detailed requirements and program design. The decision to accept, reject, or defer a requirement for implementation can occur during both the initial planning phase and the requirements analysis phase of each increment. Inputs to the development (dcut) phase of an increment are the set of requirements accepted for implementation. At the completion of the development phase, the build and integration phase begins. During this phase, all of the code is compiled and linked to produce a running system for formal testing. Problems identified during the test phase of each increment are either resolved in the current increment or deferred to a subsequent increment.

CMVC Feature and Defect processes can be applied to support the incremental development model as illustrated in Figure 2. A feature in CMVC is used to document and track new functions and design changes. A defect in CMVC is used to document and track problems and their resolution. During the initial planning phase, all documented features in CMVC that pertain to the project can be reviewed, categorized, assigned, and accepted. The accepted features in CMVC are then input to the development (dcut) phase of the increment. Problems identified are documented and tracked using the CMVC Defect. If a defect is not resolved in the current increment, then it becomes input to a subsequent increment. In some cases, resolution of a defect may require adding new function or changing the current design. In CMVC, this can be accomplished by opening a new feature and referencing the defect.

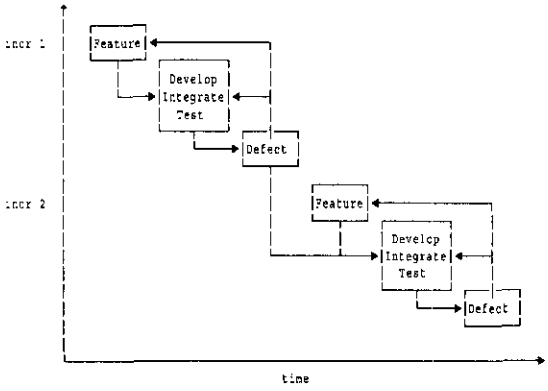


Fig. 2. CMVC Inputs to Incremental Development

An overview of the CMVC Feature process with the CMVC Track subprocess is illustrated in Figure 3. When a new feature is opened to document a new user request or a design change request, CMVC automatically notifies the owner. The owner has a choice to accept, return, cancel, or reassign the feature to someone else.

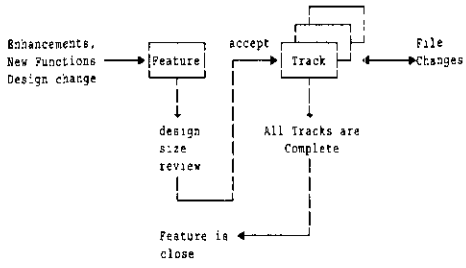


Fig. 3. CMVC Feature Process Overview

If the DSR subprocess is configured, the feature owner will need to complete the design, size, and review steps prior to accepting the feature. If the feature is accepted for implementation, then the eventual result in CMVC will be to create a new file or to make changes to an existing file that contains the code. Each file change produced by a file check-in step is monitored by a CMVC Track. If a feature affects more than one release, multiple tracks are created by CMVC to monitor the file changes. When all file changes and tracks associated with a feature have been completed, the feature is then closed.

The CMVC Defect process steps are the same as the steps in the CMVC Feature process, and therefore are not discussed here. An illustration of the CMVC Defect process is shown in Figure 4.

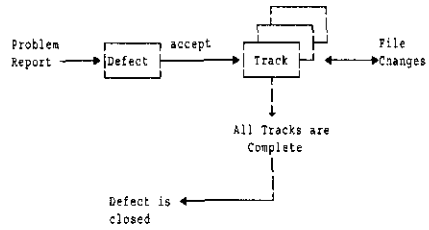


Fig. 4. CMVC Defect Process Overview

An increment or a driver in a development project can be represented by a level in CMVC. Figure 5 shows an overview of the CMVC Defect/Feature process with CMVC Track and Level subprocesses configured. CMVC Level subprocess is a configurable subprocess in CMVC, and if configured, it will monitor collection of file changes within a release. The release owner will normally define the levels in CMVC, decide which tracks to include in a level, commit the level, and complete the level. When a level is committed, all file changes associated with the level are also committed. When the level is completed, all tracks associated with the level

will be completed, and all features and defects will be closed if all of its tracks have been completed.

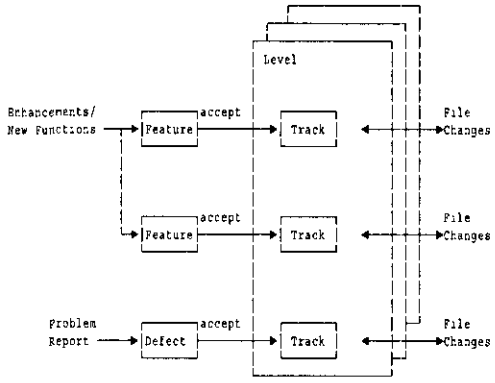


Fig. 5. CMVC Defect/Feature Process with Track and Level Subprocesses

Configuring CMVC to Match Project Size

CMVC Configurable Subprocesses

CMVC provides configurable subprocesses as shown in Figure 6. This illustration shows, in clockwise steps, an example of a feature/defect cycle from its creation to closure. Each of the configurable subprocesses is labeled and represented by a box. The arrow points to the next step in the process with the condition enclosed in parentheses. The sequence in which each subprocess plays a role can be seen in this example. Feature and defect processes are separate and independently configurable.

If no subprocess is configured, a feature or a defect can be opened and closed independently from the code changes made in the CMVC files. Also, CMVC file changes will be allowed without reference to a feature or a defect in CMVC. Configuring the track subprocess will enforce file changes by requiring a feature or a defect to be specified when a file is changed in CMVC through the file check-in action. The track will monitor the file changes for a feature or a defect by recording the file name and version number.

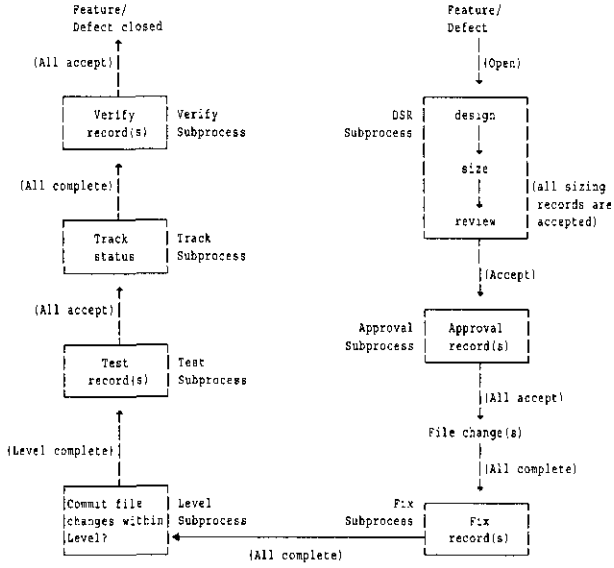


Fig. 6. CMVC Configurable Process Overview

The DSR subprocess for feature and defect can be independently configured for each component in CMVC. It is a component process. The DSR subprocess allows for design, size, and review steps. During the size step, sizing records can be created and assigned to component owners. All sizing records must be accepted to proceed to the next step. If the resolution of a feature or a defect requires coordination between component owners, then the DSR subprocess is

recommended and should be configured. DSR subprocess for feature is recommended during the development stages.

The tract, approval, fix, level, and test subprocesses can be independently configured for each release in CMVC. They are release processes. The track subprocess can be configured alone; however, the track subprocess is a prerequisite to configuring the approval, fix, level, and test subprocesses.

If the approval subprocess is configured, all approvers must accept the feature or defect to proceed to the next step. The approval subprocess is recommended for a large size project and when tight change control is required. Approvers are defined in CMVC through entries in the list of approvers.

If the fix subprocess is configured, all fix records must be complete to proceed to the next step. The fix records are automatically created when the sizing records are created in the DSR subprocess. A fix record is created per component/release and is assigned an owner. The fix subprocess is recommended, if the resolution of a feature or a defect requires coordination between component owners. The fix subprocess is recommended for a medium to large size project.

The level subprocess is recommended for a medium to large size project. If this subprocess is configured, level allows for building of increments or drivers, re-creation of a previous level, and file commits. A release owner will normally define levels within a release, decide which tracks to include, and when to commit a level.

The test subprocess is recommended for a large size project. If this subprocess is configured, test records are created and monitored. A test record is created for each tester/environment defined in the environment list. All test records must be accepted to proceed to the next step.

The verify subprocess for feature and defect can be independently configured for each component in CMVC. It is a component process. If the originator and owner of feature and defect are different, then the verify subprocess is recommended. The verify subprocess for defect is recommended during the formal testing stages.

Recommended CMVC Process Configurations

The subprocesses can be configured to meet the needs of small, medium, and large size projects. This versatile process is an advantage for development projects that grow in size over time. One CMVC server can also support multiple projects of varying sizes. Table 2 defines the project size in terms of small team (Team-S), medium team (Team-M), and large team (Team-L). The left half of Figure 7 shows the recommended configuration of release processes in relation to the project size. CMVC Release process consists of Track, Approval, Fix, Level, and Test subprocesses. The recommended configuration for a small team (Team-S) is Track subprocess only, while the recommended configuration for a large team (Team-L) is all subprocesses [5]. New named release processes can also be created with the desired combination.

The process can also be configured to meet the needs of each stage of the development life cycle: Prototype, development, test, preship, maintenance, and emergency fix. The right half of Figure 7 shows the recommended configuration of component processes in relation to current stage in the product life cycle.

TABLE 2

TEAM PROFILE

Team-S	The size of this team may be one department with a target cycle time between nine to twelve months. Each team member plays multiple roles, such as developer+architech, developer+tester, developer+builder, and developer+changeteam. There is no separate organization for test, build, and change team functions. Communication is optimum among the team members.
Team-M	This team may consist of multiple first-line organizations with a target cycle time of twelve to eighteen months. Separate teams exist for development, test, build, and change team functions. Development of releases on multiple-platforms may occur at the same time. For older products, APAR fixes, and special projects may also occur at the same time. Communication is medium among the team members.
Team-L	This team may consist of multiple second-line organizations with a target cycle time of twenty-four months Separate organizations exist for development, test, build, and change team functions. Development of multiple releases occur at the same time. For older products, APAR fixes and special projects may also occur at the same time. Communication is low among the team members.

The CMVC component process consists of design-size-review (dsr) and verify (ver) subprocesses. The component processes shipped with CMVC are named to match the product life cycle stages. The recommended configuration during the prototype stage is none, while the recommended configuration during the development stage is design-size-review for Feature (dsrFeature) [5]. When the product enters the maintenance stage, the recommended configuration is verify subprocess for Feature (verFeature) and verify subprocess for Defect (verDefect). New named component processes can also be created with the desired combination.

CMVC subprocesses are configured by selecting a named process at the time a CMVC component or a CMVC release is created. The subprocess configuration can be changed after the project has started; however, it is easier to add a subprocess than to remove it. New named release and component processes are created by the CMVC family administrator.

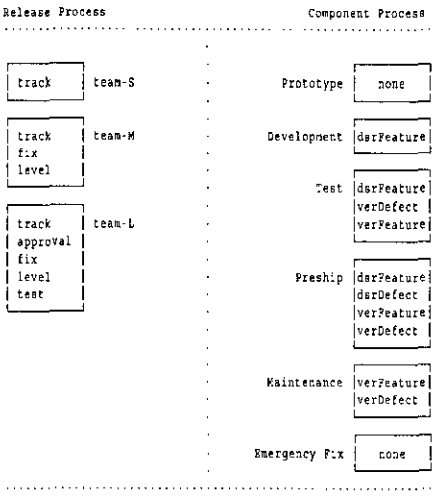


Fig. 7. Recommended CMVC Process Configurations

The named processes shipped with CMVC [6] are shown in Figure 7. The release processes shipped with CMVC are shown on the left half of Figure 7. The component processes shipped with CMVC are shown on the right half of Figure 7. The recommended process configuration can be arrived at by knowing the project size and the project stage in the life cycle.

Application Experience

Table 3 shows the actual CMVC use experience at Santa Teresa Laboratory. One project was selected from each team profile with the most CMVC experience.

Project P1 is a Team-S size project in the

maintenance phase of the product life cycle. The recommended CMVC release process configuration for Team-S size project is Track subprocess. The recommended CMVC component process configuration for Team-S size project is Verify subprocess for Defect and Feature. As Table 3 shows, project P1 has followed the recommended process configuration.

TABLE 3

ACTUAL CMVC USE DATA

Proj	Profile	# Users	Cycle Time	Life Cycle Stage	Compo-nent Process	Release Process
P1	Team-S	12	12 mo	Mainte-nance	verFeature verDefect	track
P2[10]	Team-M	50	18 mo	Test	verDefect	track level
P3[11]	Team-L	200	24 mo	Devel-opment	dsrFeature	track approval fix level test

Project P2 is a Team-M size project in the test phase of the product life-cycle. The recommended CMVC release process configuration for Team-M size project is Track, Fix, and Level subprocesses. The recommended CMVC component process configuration for Team-M size project is DSR subprocess for feature, and Verify subprocess for defect and feature. Table 3 shows that project P2 has deviated from the recommended release process configuration by not using the Fix subprocess. The role of the Fix subprocess is to coordinate file changes across multiple components to resolve Defect/Feature. Nearly all of the file changes in project P2 involved one Component, and therefore the Fix subprocess was not necessary. Project P2 has also deviated from the recommended component process by not using the DSR Feature and Verify Feature subprocesses. This deviation, however, will be corrected in the next product release.

Project P3 is starting with CMVC, and plans to follow the recommended process configuration, which is shown in Table 3.

Trademarks

Terms, denoted by an asterisk (*) in this paper, are trademarks of IBM Corporation in the United States and other countries.

PVCS, denoted by a double asterisk (**) in this paper, is a trademark of INTERSOLV, Inc.

About the Author

Seong (Steve) Yu works at Santa Teresa Laboratory in the Information Technology Services area. He is currently the site CMVC Administrator and the team leader of site CMVC Planning, Support, and User Education. He's been with the company for over 10 years, and has worked on several different products as Test Team Leader, Developer, and Product Planner.

Steve holds a B.S. degree in Mathematics-Computer Science from U.C.L.A., and M.S. degree in Computer Science from C.S.U., Chico.

[10] The DSR Feature and Verify Feature subprocesses will be configured in their next release.

[11] Project is in planning stage. The data is projected.

He can be reached at Santa Teresa Laboratory, D67/G112, 555 Bailey Avenue, San Jose, California 95141. His Internet address is syu@stlvm1.vnet.ibm.com.

References

[1] Gaitanos K., Harrison M., McNamara S., "Monitoring and Enhancing the Quality of Software Using IBM Configuration Management Version Control Tool," White Paper, IBM Toronto Laboratory.

[2] Gaitanos K., McNamara S., "Implementation of ISO 9001 Using IBM CMVC," White Paper IBM Toronto Laboratory.

[3] Ghezzi, C., Jazayeri, M., Mandrioli, D., *Fundamentals of Software Engineering*, Englewood Cliffs, NJ: Prentice-Hall, pp. 357-413, 1991.

[4] I.B.M., *Configuration Management Version Control: Concepts* Version 2 Release 1, SRL SC09-1633-00, First Edition, June 1993.

[5] I.B.M., *Configuration Management Version Control Server Administration and Installation*, Version 2 Release 2, SRL SC09-1631-01, Second Edition, December 1993.

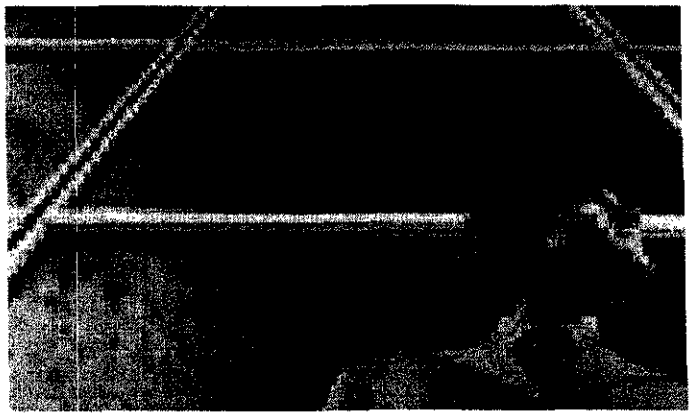
[6] I.B.M., *Configuration Management Version Control: User's Reference*, Version 2 Release 2, SRL SC09-1597-01, Second Edition, December 1993.

EXHIBIT 5

IBM e customer relationships SIEBEL.

IBM takes customer service to the next level:

The world's largest implementation ever of Siebel *e*Business Applications continues



ibm-siebel.com

Appendix C:
Version control process and tools in IBM CRM

The development of the IBM CRM solutions is a complex undertaking with numerous moving parts. To manage this process with accuracy and control, a number of processes and tools are employed. The situation would be complex enough if one were only managing the changes needed to customize the Siebel applications. There are, however, definitions, code, parameters and numerous other items that must be created and maintained for all of the application's various components.

The major technology component of this mix is the use of an IBM configuration management repository and version control (CMVC) as a change management repository that allows IBM to track and coordinate the myriad pieces required in a release. Employing CMVC in the Siebel context is a challenge in itself as the Siebel Tools application is used to manage the coding and specifications of all the various changes to parts of the Siebel *e*Business Applications and servers.

Fortunately, Siebel Tools allows the definition of "projects" within the application, which defines a specific set of "pieces" that can then be imported and exported in a .sif file format. IBM uses CMVC to version control the various .sif files as a means of managing the change process with Siebel Tools. Today, this means creating and changing a component with Siebel software, exporting the .sif, and checking the .sif into CMVC as a part. Further changes require the .sif to be checked out of CMVC, imported into Tools, updated, exported and checked in. IBM is working with Siebel Systems to create a more direct interface to check projects directly in and out from CMVC.

In addition to Siebel *e*Business Applications components, IBM also has parts defined for all of the various infrastructure and system components needed in the complete IBM CRM solutions. This can be as simple as a set of queue name definitions for MQSeries all the way to a copy of the correct install package, instructions and configuration for a system component such as UDB.

In addition to revision control, CMVC has a very sophisticated defect tracking process, which allows IBM to work on and correct defects on any of the parts managed in the repository. IBM also supplements the CMVC repository facilities with a "build process." The build process is a set of scripts and programs that can obtain a set of parts at a particular level and create all the deployment artifacts necessary to deploy a usable instance of the IBM CRM application system. This includes creating a valid definition file, or .srf, as well as the other artifacts needed by Siebel *e*Business Applications. This process is automated to the extent that the sandbox-level test systems can automatically be rebuilt each night, incorporating the fixes and changes performed during the previous day.