LA-UR-84-2335

CONF-8[illegible]

LA-UR--84-2335

DE84 015515

TITLE CONFIGURATION MANAGEMENT FOR MISSION-CRITICAL SOFTWARE: THE LOS ALAMOS SOLUTION

AUTHOR(S) G. Cort and D. M. Barrus

SUBMITTED TO Softool Users Group Meeting, Santa Barbara, CA, September 10, 1984.

[illegible]

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

Configuration Management for Mission-Critical Software: the Los Alamos Solution*

G. Cort and D. M. Barrus
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

Introduction

As has been the case for most of the disciplines of software engineering, the evolution of the principles of software configuration management has been driven mainly by the requirements of large-scale software development projects. Although this situation has resulted in very effective and efficient strategies for managing these giant projects, the very different needs of small or intermediate sized projects have been largely ignored. This has served effectively to deny both the immediate and the long term benefits of software engineering in general, and software configuration management in particular, to the majority of software development projects. Far more serious is the dangerous attitude fostered by the large scale approach to these very important disciplines, namely that the techniques of software engineering and configuration management can only be cost-effective when applied on a grand scale.

In this paper we present our experiences as a small group responsible for the development of a moderately large real-time data acquisition system. During the early stages of our project we recognized the need for a rigorous software configuration management system to support our development and maintenance activities. This paper describes our approach to the utilization of the Softool Change and Configuration Control (CCC) environment. The steps that we have taken to develop a very powerful development/configuration management environment (incorporating CCC) are outlined and justified. The extension of the Los Alamos system to management of large-scale projects is discussed.

Project Organization

In order to establish the requirements and operational constraints which led to the development of the Los Alamos system, a brief description of our facility and the organization of our project is appropriate. The Los Alamos Weapons Neutron Research Facility (WNR) is a world-class neutron scattering installation devoted to basic research in physics, chemistry, materials science and biology. Operating in conjunction with the 800 MeV linear accelerator at the Los Alamos Meson Physics Facility (LAMPF) the facility supports an expanding, international user community. A major facility upgrade currently being implemented will significantly enhance present capabilities and will transform WNR into one of the world's premiere neutron scattering centers.

The same upgrade, however, will render the existing real-time data acquisition system obsolete. Its replacement, which is currently under development by the Computer Section of the WNR Operations Group, will ultimately consist of a network of 8-12 computers of the VAX 11/750 class each hosting the VMS operating system. Each computer will be dedicated to acquiring data from a single spectrometer. To accomplish this task, each computer will execute identical data acquisition software.

The projected size of the software system being developed to meet the data acquisition requirements of the new facility is approximately 150 K executable lines. The programming language chosen for the project is an extended version of Pascal. Reliable operation of this system is essential as software failures can result in total disruption of the operation of the facility. Because of the great expense incurred in producing the neutron beam, and the high demand by users for access to the facility, the economic, political and scientific consequences of a system failure can be quite serious. Because of the sheer size of the project, the complexity of the software being developed, and the mission-critical nature of the system, it was decided during the early stages of the project to employ a rigorous software engineering approach, including the incorporation of stringent software configuration management.

In addition to the basic hardware and software facility characteristics presented above, the organization and structure of the software development staff is extremely important in determining the level and mode of configuration management appropriate for the project. Our organization consists of three very senior staff members with full-time responsibility for software design and implementation. In addition, we have available the equivalent of approximately two full-time people to support the development effort. These individuals range in experience from very senior staff members with partial responsibility for software to junior programmers and data analysts.

The project management structure is also quite different from that associated with most large-scale development efforts. The small staff attached to the project does not warrant the multilayered, highly stratified management structure imposed on large development projects. Indeed, a single manager oversees the entire software development effort. Heavy reliance on the experience and judgement of the software staff further reduces management visibility to a minimal level.

Advantages of the CCC Environment

After evaluating various commercial configuration management systems, we chose the CCC as the tool best fitted to support our configuration management effort. The CCC environment can provide virtually unbreachable security for system sources (and documentation), thereby eliminating the possibility of inadvertent or unauthorized modification of any of these key system components. This capability is of particular importance in a highly volatile development environment such as ours: one in which every programmer has access to system management resources and therefore is potentially capable of bypassing all file protections established by the operating system.

The CCC environment also provides us with a comprehensive, automated version control system, a feature that is essential to the conduct of an effective configuration accounting effort. This feature gives us the capability to define the precise configuration of any software component of the data acquisition system. In addition, it provides for fallback configurations that can be utilized in the event of a serious failure of a primary software component, thus allowing the data acquisition task to continue (though possibly with reduced capability) while the primary component is under repair.

The almost unlimited extent to which the CCC macro facility allows the configuration management environment to be automated is another extremely valuable feature. This capability is particularly attractive within the context of our project for which the relatively small size of the technical staff demands that the overhead associated with support functions (such as configuration management) be kept to an absolute minimum. The introduction of automated procedures into the environment not only decreases the time spent on configuration management functions, but also significantly enhances the accuracy and reliability of all transactions.

Finally, the capability to minimize the size of the configuration data base by defining global text structures for parent configurations provides an effective mechanism for conserving precious disk resources. This allows many generations of each software component to be maintained in the data base without duplicating redundant information, thereby eliminating the necessity of restoring a previous version from secondary storage in the event of a maintenance emergency. This feature significantly enhances the ability of the configuration management staff to react to emergency situations as they arise.

The Need for an Extended Environment

Although we recognize the CCC environment as an extremely powerful tool to support configuration management activities, it is our position that the conventional methodologies for utilizing this tool are not adequate to meet the needs of a small software development project such as ours. At the extreme of maximum CCC utilization, the methodology requires that every development and maintenance programmer work entirely within the confines of the CCC data base, using CCC commands and the CCC editor to accomplish all programming and maintenance activities. At the opposite extreme, programming and maintenance staff do not interact with the CCC data base, but instead conduct their programming activities externally. A manager is then responsible for copying all work from the users' environments into the CCC data base at regular intervals. The deficiencies of these methodologies are discussed below.

The policy of maximum CCC utilization allows management to exercise a high level of visibility throughout the development process, and management resources and therefore is potentially capable of bypassing all file protections established by the operating system.

The CCC environment also provides us with a comprehensive, automated version control system, a feature that is essential to the conduct of an effective configuration accounting effort. This feature gives us the capability to define the precise configuration of any software component of the data acquisition system. In addition, it provides for fallback configurations that can be utilized in the event of a serious failure of a primary software component, thus allowing the data acquisition task to continue (though possibly with reduced capability) while the primary component is under repair.

The almost unlimited extent to which the CCC macro facility allows the configuration management environment to be automated is another extremely valuable feature. This capability is particularly attractive within the context of our project for which the relatively small size of the technical staff demands that the overhead associated with support functions (such as configuration management) be kept to an absolute minimum. The introduction of automated procedures into the environment not only decreases the time spent on configuration management functions, but significantly enhances the accuracy and reliability of all transactions.

Finally, the capability to minimize the size of the configuration data base by defining global text structures for parent configurations provides an effective mechanism for conserving precious disk resources. This allows many generations of each software component to be maintained in the data base without duplicating redundant information, thereby eliminating the necessity of restoring a previous version from secondary storage in the event of a maintenance emergency. This feature significantly enhances the ability of the configuration management staff to react to emergency situations as they arise.

The Need for an Extended Environment

Although we recognize the CCC environment as an extremely powerful tool to support configuration management activities, it is our position that the conventional methodologies for utilizing this tool are not adequate to meet the needs of a small software development project such as ours. At the extreme of maximum CCC utilization, the methodology requires that every development and maintenance programmer work entirely within the confines of the CCC data base, using CCC commands and the CCC editor to accomplish all programming and maintenance activities. At the opposite extreme, programming and maintenance staff do not interact with the CCC data base, but instead conduct their programming activities externally. A manager is then responsible for copying all work from the users' environments into the CCC data base at regular intervals. The deficiencies of these methodologies are discussed below.

The policy of maximum CCC utilization allows management to exercise a high level of visibility throughout the development process, and provides the capability to identify software version changes with an extremely fine time resolution. Unfortunately, this approach also imposes severe overheads on both configuration management and development personnel. The most severe management overhead derives from the necessity for the data base administrator to define and maintain access control information for every CCC user. This problem is further complicated by the extremely volatile development environment that is often associated with small projects: users' access control information may require modification on a daily or even hourly basis. Add in a constraint that requires all maintenance operations to be performed in a modular fashion (programmers are allowed access to only those modules of a software component that actually require modification) and the process of maintaining access control information becomes increasingly error-prone and time intensive. It should be noted that this activity cannot be extensively automated, so there is little hope of reducing these overheads through the use of the CCC macro facility.

Also, the maximum utilization strategy imposes intolerable overheads on the technical staff. The effective relocation of the development environment to within the confines of the CCC data base has the immediate consequence of making standard development tools (compilers, linkers, etc.) as well as locally developed automated software support tools inaccessible to the developer. As a result, what should be a simple compile-link procedure becomes tedious, time consuming, multistep operation involving exportation of the appropriate modules from the data base, performance of compilation and link steps in the host operating system environment, and importation of the source modules back into the data base. In addition to the direct deleterious effects upon developer productivity, the imposition of such overheads can foster resentment and can result in serious erosion of morale within the technical staff. To support a modular maintenance effort within this environment becomes even more difficult, requiring a significantly increased level of participation by the data base administrator.

Additional unacceptable overheads are also characteristic of the maximum utilization implementation. Developers are required to become proficient with new software interfaces in order to operate within the CCC data base. In some cases these new interfaces may be perceived as less effective than tools that exist at the operating system level (for instance, programmers resist abandoning the versatile, full-screen VMS EDT editor for the less powerful, line-oriented CCC editor). Additionally, response times deteriorate rapidly as more users are forced to access the data base simultaneously. Coupled with the extra response time overhead introduced by a policy of archiving incremental changes for most recent versions, these delays can seriously degrade development productivity.

The minimum utilization methodology also presents serious problems as a configuration management implementation strategy. Although access control, tool accessibility and response time overheads are largely eliminated by this approach, significant new management overheads are introduced. Foremost among these is the increased effort required to export modules from the CCC data base for maintenance, especially in a modular maintenance environment. When used in this mode, the CCC environment seems to be reduced to an extremely sophisticated (and expensive) backup utility.

Both methodologies seem to allow the CCC data base to become cluttered with uncertified intermediate software versions. This generally results in rapid increase in data base size and decreased intervals between data base maintenance and backup activities. Almost regardless of the time resolution associated with the the smallest increment of change, the benefits to be gained by saving uncertified versions in the data base are offset by the increased maintenance burden placed on the data base administrator.

In short, there seems to be a basic incompatibility between environments that promote a strong development effort (the VMS operating system environment, for example) and those, such as CCC, that support rigorous, automated configuration management activities. Environments in which developers thrive present severe difficulties for configuration management personnel. The converse also appears to be true. Conventional approaches to the resolution of these problems generally force one of these groups to work within an inadequate environment in order to preserve the effectiveness of the other group. In worst case situations, each group is forced to endure a compromise solution in which both parties sacrifice significant capabilities and no one is satisfied. The Los Alamos approach, however, is to define a new methodology that completely isolates development activities from the configuration management effort, thereby allowing the full power of each environment to be exploited to its fullest. The unique feature of this strategy is the provision of an interface between the two environments that allows for automated interaction between them, and actually melds them into a single, comprehensive hybrid environment for software development and configuration management.

The Hybrid Environment: Specifications

Within the context of our project, the following properties were identified as required features of the hybrid environment and its associated configuration management methodology:

User exclusion from the CCC data base. All development/maintenance activities must be conducted within the VMS host operating system environment. This requirement was specified in order to eliminate the management and developer overheads associated with CCC data base transactions and maintenance. Only the CCC data base administrator is permitted access to the data base.

Only certified software is maintained under configuration control. Only software that has been reviewed and passed by the facility Configuration Control Board (CCB) is accepted into the configuration management environment. Likewise, specific approval of the CCB is required before any software is released from configuration control (by transfer to the development environment). All uncertified software versions (generally intermediate versions of modules undergoing maintenance or development) remain in the development environment.

Reliance is placed upon ordinary facility software backup procedures to provide adequate capability for reconstruction of modules in the development environment.

<u>The hybrid environment must impose no additional overheads upon the developer.</u> All configuration management tasks must be the exclusive responsibility of configuration management personnel. In addition, there must be no degradation in system response attributable to the hybrid environment.

<u>The hybrid environment must support the automation of virtually every configuration management task.</u> Because the configuration management staff is responsible for all aspects of the configuration management effort, and because these staff members generally have significant development responsibilities as well, automated procedures must be available to reduce the effort and increase the reliability of all configuration management transactions.

The Hybrid Environment: Implementation

The hybrid environment is comprised of a development environment and a configuration management environment, each of which is strictly isolated from the other. The development environment consists of the VMS operating system utilized in the conventional manner and partitioned into the usual user accounts and directories. All activities that take place within the development environment are the exclusive responsibility of the software developer and are not monitored or influenced in any manner by the configuration management staff.

The configuration management environment consists of a Configuration Data Base (CDB) and automated procedures (VMS command files and CCC macros) to operate on CDB elements. The configuration management staff is responsible for performing all operations on the CDB.

The organization of the CDB reflects our operational requirement that modules be maintained at different levels of configuration control depending on the function, utilization and current change processing status of a module. To meet this requirement, the CDB is divided into a Class 1 and a Class 2 partition. The Class 1 partition consists of the CCC data base and is intended to hold modules for which access must be restricted to configuration management personnel only. Source code, user's documentation and test results are examples of modules that must be maintained under Class 1 configuration management. The Class 1 partition is organized such that each constituent CCC SYSTEM structure is devoted to a different data acquisition subsystem (primary commands, tools, utilities), and each CONFIGURATION structure subordinate to a given SYSTEM represents a unique version of one of the computer programs that comprise the subsystem. MODULE structures are used to functionally subdivide each version into user's documentation, source code and test report categories. Subordinate TEXT structures comprise the constituents of each category. Only the CCC data base administrator is authorized to access structures that reside within the Class 1 partition.

The Class 2 partition consists of a hierarchy of protected VMS directories that contain software modules that must be accessed on an on-demand basis by data acquisition system users or development personnel. Included in the list of Class 2 modules are executable images, libraries of object modules and support data bases. All users have read-access to Class 2 modules; only configuration management personnel have modify-access. Obviously, modules that exist in the Class 2 partition are not as secure as those in the Class 1 partition. However, because these modules exist in non-ASCII format, and because any Class 2 module can be simply rebuilt (usually by a compile or link operation) from one or more Class 1 modules, the reduced security is not considered a serious problem.

The CDB structure is primarily responsible for reducing developer overheads to levels comparable to those that would exist in the absence of any configuration management activities. By extensively automating the CDB (using CCC macros for the Class 1 partition and VMS command files for the Class 2 partition) management overheads can also be drastically reduced. Indeed, the only aspects of the configuration management process that do not lend themselves to automation under the hybrid environment are those of 1) releasing software from the Class 1 partition into the development environment and 2) admitting software (after certification) from the development environment into the CDB. This inability to integrate the constituent environments in an automated fashion was considered a serious deficiency of the hybrid approach in light of the fact that these processes comprise the vast majority of the activities of the configuration management staff.

Integrating the Environments

In order to address this deficiency, we further extended the hybrid environment by defining an interface data structure that enables the complete automation of software transfers between the constituent environments. The structure is called a Program Source File List (SFL) and consists of a text file that describes the structure of a program. Each program that is maintained in the Class 1 partition has a corresponding SFL that resides with the program source code in the appropriate CONFIGURATION structure.

An SFL consists of a list of each software module that must be compiled/linked to build the executable image for a particular computer program. The SFL is organized with one module name per line and allows commentary material to be included after any module name. The SFL also contains information that defines the status (unmodified, modified, or new) of each module in the list. A sample source file list is shown below.

Source File List for Program ADD :

```
ADD                     ! Main program
VALDATBAS               ! Data base validation routine
SEEKENTRY               ! Entry locate routine
UPDATE                  ! New entry addition routine
PMPTUSER                ! General prompting routine
PARSELINE               ! Command decoding routine
```

In order to demonstrate the degree to which SFL's support the unification of the hybrid environment through automation of the interface between the constituent environments, we present the following example of a simple maintenance operation. Consider the ADD program for which the SFL is presented above. Consider also that a software fault associated with the execution of ADD has been identified and reported. Analysis indicates that bugs exist in the SEEKENTRY and the PARSELINE subprograms and (in accordance with the modular maintenance policy) a request has been placed with the configuration administrator to release these modules into the development environment (i.e. into the maintenance programmer's local VMS directory). Using manual procedures to accomplish the appropriate transfers from the CCC data base to the maintenance programmer's VMS directory is a tedious and error-prone operation. This is especially true in light of the fact that a transfer operation must be performed on every module of the computer program, regardless of how many modules are to be modified. (This results from the fact that object modules must be generated for all modules that are not subject to modification, and these object modules must be transferred to the Class 2 partition in order to allow the ADD program to be linked prior to testing.)

By utilizing the ADD source file list, however, the entire manual process described above can be replaced by an automated procedure that reduces the overhead imposed upon the CCC data base administrator to trivial levels. The only step performed manually involves editing the SFL to indicate which modules are to be transferred to the maintenance programmer. This is accomplished by editing the appropriate SFL (with the CCC editor to place an asterisk (*) before the name of each module to be transferred. Within the context of this example the edited SFL for the ADD program would appear as follows:

Edited Source File List for Program ADD :

```
  ADD                   ! Main program
  VALDATBAS             ! Data base validation routine
* SEEKENTRY             ! Entry locate routine
  UPDATE                ! New entry addition routine
  PMPTUSER              ! General prompting routine
* PARSELINE             ! Command decoding routine
```

A CCC macro is then invoked that parses the edited SFL and transmits the source code for the flagged modules to the appropriate maintenance account, and sends object modules for all other SFL entries to the Class 2 partition from where they can be accessed by the maintenance programmer at link time. In addition, the macro sends a copy of the edited SFL to the maintenance account.

Within the development environment, the SFL can also be utilized to streamline the job of the developer or maintenance programmer. To demonstrate this let us continue our example by assuming that appropriate modifications have been made to SEEKENTRY and PARSELINE. We will also assume that the programmer has decided that, in addition to these modifications, an entirely new module (called VALCOMMND) is also required and has been developed.

The maintenance programmer is now prepared to recompile all of the modified modules and the newly developed module prior to relinking the ADD program. This could be done manually, or even with a command file written and maintained by the programmer. A far simpler approach is to use the information contained in the SFL as input to an automated utility (a VMS command file) that recompiles all modified or newly developed modules. Prior to invoking this utility, the programmer must re-edit the SFL to indicate any newly developed modules associated with the program. This is accomplished by flagging the names of all newly developed modules with two asterisks and adding them to the SFL. For this example the re-edited SFL would appear as follows:

Re-edited Source File List for Program ADD :

```
   ADD                   ! Main program
   VALDATBAS             ! Data base validation routine
 * SEEKENTRY             ! Entry locate routine
   UPDATE                ! New entry addition routine
   PMPTUSER              ! General prompting routine
 * PARSELINE             ! Command decoding routine
**VALCOMMND              ! Command validation routine
```

The utility parses the SFL and compiles any module that is flagged as new (**) or modified (*).

Similar support can be provided for the link activity. An automated procedure can be supplied that parses the SFL and retrieves each required object module from one of several locations depending on the status (new, modified or unmodified) of the corresponding entry in the SFL. Objects for new and modified modules are linked from the maintenance account; objects for unmodified modules are linked from the Class 2 partition.

In addition to reducing the overheads imposed upon the maintenance/development programmer, utilization of these standard compilation and link tools guarantees that the same set of compilation and link options are used in every operation. This promotes a level of software uniformity that would be difficult to obtain with manual procedures.

The final step in the maintenance cycle for program ADD involves readmitting (after certification) the modified and newly developed modules to the Class 1 partition (CCC data base). Depending upon local configuration management standards and upon the level of maintenance performed, this step may also require generation of a new CONFIGURATION data structure within CCC to accomodate the modified software. To accomplish this task a CCC macro can be invoked to insert the new CONFIGURATION, import and parse the SFL, and import all modules that are flagged within the SFL as new or modified. As a final step, the macro deletes all status flags from the SFL. Again, virtually all manual procedures are eliminated from what would otherwise be a very complex task.

In addition to eliminating the tedium and significantly reducing the time involved in processing new and modified modules, use of SFL-based automated procedures and utilities at all levels of the development and configuration management efforts virtually eliminates the possibility of corrupting the Class 1 partition due to an error or oversight on the part of the developer or the configuration management staff. Configuration management efforts that rely upon manual procedures to update a data base of protected software are susceptible to admitting uncertified modules to the data base, or failing to admit all of the new or modified modules for a program to the data base. In either case, if these errors are not <u>immediately</u> detected and rectified, the integrity of the data base can be seriously compromised. By providing SFL-based tools that are used by both the development/maintenance and the configuration management communities, however, one can guarantee that <u>all</u> modules and the <u>same</u> modules that comprise a (successfully tested) program are readmitted to the data base.

The final aspect of SFL utilization that we will present is the application of SFL's to the automation of software system rebuilds. Within the context of our system, the term <u>system rebuild</u> denotes a process whereby all software subordinate to a particular data structure in the Class 1 partition (CCC data base) is recompiled and relinked, and the appropriate compone ts of the Class 2 data base are updated (with the new executable images, for example). To perform this task manually, even for a very small system, can be an enormously complex and time-intensive undertaking.

By utilizing the information within source file lists, however, this process can be completely automated. A CCC macro is invoked to modify the SFL for each program in the data base, flagging each constituent module for transfer out of the Class 1 partition. This macro then invokes the software release macro (discussed above) to transfer all source modules and the corresponding SFL's to a location in the Class 2 partition from which they can be compiled and linked. The standard compilation and link utilities can then be executed from a command file to accomplish the recompilation, relink and recataloging of the resulting executable image for all exported software. All sources and objects are then deleted from the Class 2 partition. In this manner the entire system can be rebuilt extremely quickly and reliably.

Configuration Accounting within the Hybrid Environment

One of the premiere difficulties associated with the conduct of a program o configuration management relates to the generation and control of large amounts of printed material. Especially for small projects, and regardless of the degree to which interface overheads are reduced by the utilization of SFL's, the effort required to generate, update, file and retrieve the printed byproducts of configuration management activities often dwarfs the savings gleaned from utilizing automated interface procedures. For a project of our size and organization, the most significant contributor to this sea of paper is the configuration accounting effort.

Our configuration accounting procedures specify the use of four different reports to initiate software changes and track the change processing status of a software component through the development, maintenance and certification processes. Typical of these forms is the Discrepancy Report (DR), a standard form that is completed and submitted by system users to report a software fault and to initiate the maintenance activity to repair it. In order to track these reports accurately, it is necessary to maintain logs of pending (unassigned), in-progress, and completed DR's as well as a chronologically-organized log for all DR's. In addition a copy of each DR submitted against a particular software component should be filed (in the Class 1 partition) with the source code of the repaired software. Obviously, maintaining the various logs of printed copies of these forms requires a major clerical effort. In addition, even if the logs can be maintained in good order, the process of generating reports that summarize their contents cannot be easily accomplished with manual procedures.

Within the structure of the hybrid environment, however, these problems can be very effectively addressed by automating all aspects of the configuration accounting process. Templates for all configuration accounting forms reside in the Class 2 partition of the CDB, where they can be accessed by all users. A template can be completed using a text editor and submitted via electronic mail to a special holding area in the Class 2 partition where it is assigned a unique identifying number. A collection of CCC macros can then be used to perform all operations upon a submitted DR, including:

a) importation from the holding area to the chronological log (within the CCC data base);
b) modification of the DR text to record the identifying number on the DR;
c) assignment of a change name to reflect the PENDING status of the DR;
d) automatic change of status of the DR from PENDING to ASSIGNED when the DR is assigned to a programmer or analyst. Upon assignment, automatic exportation of the DR to the programmer's development environment;
e) automatic change of status of the DR from ASSIGNED to COMPLETED when the maintenance specified by the DR has been completed. Upon completion, automatic exportation of the satisfied DR to the <u>originator's</u> account (in the Class 2

partition), and automatic copying into the Class 1 data structure that contains the certified, repaired software sources.

In addition to supporting automated operations upon configuration accounting forms, the hybrid environment, by making use of CCC's data base management capabilities, supports automated management of these forms. CCC macros that employ the LISTCHANGE and LISTSTRUCTURE commands can be used to generate the following reports on the status of the change processing effort:

a) names and status of all entries in the chronological log ;
b) all change processing requests that have been submitted after a specified date and time ;
c) individual lists of all change processing requests corresponding to a particular status: PENDING, ASSIGNED, or COMPLETED.

For a small project with a limited staff, the hybrid environment makes possible the implementation and support of very powerful automated change processing and configuration accounting procedures without burdening the development/maintenance communities with additional overheads. Definition of these procedures accomplishes the goal of addressing all configuration management activities in an automated fashion, thereby maximizing the effectiveness and productivity of the configuration management staff without interfering with the development/maintenance effort. It also allows for electronic storage maintenance, transfer and retrieval of information that would otherwise be maintained in printed form, thereby moving us one step closer to the goal of "paperless" project management.

Conclusions

In this paper we have presented the results of our efforts to implement an effective automated configuration management environment to support a small software development project. We have demonstrated that the introduction of a hybrid environment that exploits features of the Softool CCC (for configuration management support) and the VMS operating system (for development support) provides an extremely powerful structure within which both of these complementary activities can be conducted. We have further demonstrated that a simple interface data structure (the SFL) can be defined that allows automation of the interactions that must take place between the constituents of the hybrid environment. Finally we have shown that the performance of this system (in terms of operational overheads, convenience and user training) significantly exceeds that of conventional configuration management environments. Indeed, we have shown that the capabilities of a developer operating within the hybrid environment are actually enhanced, without imposing any significant additional overheads.

The hybrid environment approach to configuration management was developed specifically to address the requirements of a small software development project. These requirements dictated the elimination of intrusions upon the development effort by configuration management activities. In addition, the ability to automate all phases of the configuration management effort was deemed the only practical way to guarantee that all configuration management activities could be carried out by a very small staff. It is our opinion, however, that the hybrid environment approach is also appropriate for use in conjunction with large-scale software development projects. Although the high level of management visibility supported by the hybrid environment, and the prohibition against retaining uncertified versions within the CCC data base may be considered limitations, the tremendous reduction in the overheads imposed upon both management and technical staffs could potentially result in even greater productivity gains than are seen on a small project.

We predict that the minimum impact of the implementation of the hybrid development/configuration management environment upon a large-scale software development project would be the re-assignment of a large fraction of the configuration management staff from tedious manual tasks to (more productive) development-oriented activities. Certainly, it is true that the automated procedures that we have described in this paper constitute a minimum exploitation of an extremely powerful resource: a subset that enables the small project to conduct effective configuration management. The enhancements to this system that might be realized by redirecting the efforts of staff formerly engaged in manual configuration management activities to the development and support of new automated capabilities could revolutionize this very important software engineering discipline.

* This work was performed under the auspices of the U. S. Department