LEGIBILITY NOTICE

A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

1

LA-UR- 88-1988

Received by OSTI

AUG 0 4 1988

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

TITLE: THE LANSCE SOFTWARE MANAGEMENT ENVIRONMENT

LA-UR--88-1988

DE88 014436

AUTHOR(S): Steve Donahue and Gary Cort, LANSCE

SUBMITTED TO: PROCEEDINGS OF THE 1988 DECUS SYMPOSIUM, MAY 16- 20, 1988, CINCINNATI, OHIO

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes.

The Los Alamos National Laboratory requests the publisher identify this article as work performed under the auspices of the U.S. Department of Energy.

Los Alamos

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

MASTER

FORM NO 836 R4
ST NO 2629

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

THE LANSCE SOFTWARE MANAGEMENT ENVIRONMENT

Steve Donahue and Gary Cort
Los Alamos National Laboratory
Los Alamos, New Mexico

ABSTRACT

We present the details of an innovative software management environment implemented to address life cycle software support requirements at the Los Alamos Neutron Scattering Center (LANSCE). The system provides an automated configuration management framework within which all software development and maintenance activities are performed.

The LANSCE environment provides a comprehensive configuration management (CM) system based upon a commercially available CM tool. The CM component ensures the integrity of all LANSCE application software, automatically performs status accounting operations, and provides a flexible automated facility for rebuilding resident software components and restoring them to the baseline. It employs high-level data structures both to enhance significantly the level of automation that can be achieved and to provide extensive, integrated support to the software development community.

We discuss the software management philosophy that led to the development of the LANSCE environment on programmer productivity, system integration and software management effectiveness.

INTRODUCTION

The Los Alamos Neutron Scattering Center (LANSCE) presents unique challenges to the development, maintenance, and management of the software upon which it relies for its operation. The LANSCE mission is to provide a state-of-the art neutron scattering facility to support forefront experiments in the disciplines of physics, chemistry, materials science, and biology for an international community of users.

The computer support required to accomplish this mission is formidable. Each of the six neutron scattering instruments is supported by an identical, locally-developed, high-speed data acquisition computer that is capable of processing in excess of 2 million neutron scattering events per second. Additionally, each instrument has the exclusive use of a dedicated microVAX II/GPX workstation to interface with the custom hardware, manipulate and display data, and perform further analysis and refinement of the experimental results. These systems reside on a baseband network that allows them to communicate with each other as well as with LANSCE support computers for data analysis, data archiving, software development, software configuration management, and network system management.

The LANSCE Software Section is responsible for the development, maintenance, operation, and management of all computer software[1] required to support these systems. This software is necessarily mission-critical in nature. The LANSCE duty cycle specifies six months of virtually uninterrupted operation followed by six months of maintenance and preparation for the next operation cycle. Coupled with the facts that the LANSCE neutron beam is exceedingly expensive to produce, that a LANSCE scientist may have available only several hours (scheduled months in advance) to complete an experiment, as well as the significant expense incurred in sample preparation and travel to the LANSCE facility, software failure that results in instrument unavailability has serious scientific, economic and political consequences.

As a result, the capability to certify the functionality of all LANSCE software and to guarantee the integrity of all certified software components is crucial to the viability of the LANSCE facility. Our approach has been to institute a rigorous and comprehensive software development methodology based upon sound software engineering principles, to test exhaustively and formally all software prior to certification, and to implement a software management strategy that addresses the four classical aspects[2] of software configuration management: Configuration Identification, Configuration Control, Configuration Status Accounting and Configuration Auditing.

Whereas most large projects can devote significant resources to the support of such an approach, the LANSCE Software Section is able to bring only very limited resources to bear. The section comprises one manager,

three full-time software engineers, several part-time programmers, and a part-time Configuration Manager. No separate organizations exist to provide quality assurance or software management support. All aspects of the development, maintenance, certification and management of the LANSCE software are the responsibility of the members of the software section.

The large amount of software to be managed further complicates the situation. Existing software comprises more than 2000 source code modules organized into approximately 200 distinct computer programs and approximately 150 library modules. The entire system is estimated at roughly 250,000 lines of source code with an additional 750,000 lines of supporting documentation. Furthermore, enhancements and new developments are expected to increase the size of the system threefold over the next five years.

For such a small section to deal effectively with so large a software system, we require methodologies that are highly automated and that impose extremely low overheads on our precious personnel resources. Previous papers[3-7] have presented our solutions in the software development and certification arenas. This work describes the automated, high-performance software configuration management (SCM) system that we have implemented to support the software management function.

THE LANSCE SOFTWARE ENGINEERING ENVIRONMENT

At LANSCE, software is developed according to an evolutionary, life-cycle methodology that emphasizes rigorous standardization, strict modularity, comprehensive documentation, and exhaustive formal testing. As part of any development (or maintenance) cycle, each of four baselines is developed in sequence: the specification, preliminary design, detailed design and implementation baselines. Each baseline consists of well-defined components, all of which must be completed and certified prior to start of work on the next baseline in the sequence. Baseline certification is the responsibility of the LANSCE Configuration Control Board (CCB), which reviews extensively each baseline for completeness, compliance with facility standards, and functionality.

The CCB exerts a powerful influence on the conduct of software development at LANSCE. Comprised of all of the members of the Software Section, as well as a user (scientist) representative, the CCB conducts in-depth peer reviews to assess the inherent quality of all software baselines, thereby exercising complete authority over the software certification process. The role of the CCB, however, encompasses a much wider scope than software peer reviews. The CCB has responsibility for authorizing any and all changes to the LANSCE software system. This responsibility extends to evaluating work requests, granting access to certified software for modification or enhancement, and authorizing software rebuild or rebaseline activities as well as the previously described software certification duties. Consequently, decisions that may impact the overall integrity of the LANSCE software system are made by the CCB, rather than by any individual developer.

As an example of the system in operation, consider a typical software maintenance (bug fix) cycle. The cycle begins with the receipt by the Configuration Manager of a user-generated Discrepancy Report (DR) that describes the problem. The DR is submitted to the CCB at its next meeting, where it is assigned to an analyst for evaluation. The analyst determines whether the DR has merit (*e.g.* whether the problem is repeatable) and the scope of the required remedial effort, including identification of which modules of the affected computer program require modification. The CCB then prioritizes the DR and authorizes the release of the affected modules to a developer.

Upon receiving authorization from the CCB, the Configuration Manager initiates the release process. Source code is released to the developer only for those modules that require modification. Object modules are made available for the remainder of the software. This policy guarantees that no unauthorized or inadvertent changes are made to modules that do not require modification. In addition, the developer is not overwhelmed by a mass of source code that is irrelevant to the task at hand.

The developer then proceeds to update the relevant baselines according to the requirements of the work request. For example, if the DR results in a change in software requirements, the specification baseline is updated first, followed by changes (as required) to the preliminary design, detailed design and implementation baselines. Each baseline is reviewed upon completion by the CCB, although for particularly simple tasks the CCB may replace this series of reviews with a single review at completion of work.

A comprehensive suite of software tools is furnished the developer to assist in all aspects of the software repair activity. Tools are available to support the modification of the documentation that comprises all software baselines. Additional tools streamline the process of recompiling and relinking the software system so that it is unnecessary to develop complex command procedures to perform these mundane tasks.

The capability is also provided for the developer to make available uncertified (prerelease) versions of the repaired software to a subset of users without impacting the overall user community. This facility is particularly useful for implementing emergency fixes or enhancements that impact only a small number of users. For example, a user may encounter a need for a particular software enhancement to complete an experiment successfully. The enhancement may be of no interest to anyone else in the user community, but is crucial to the user who requested it. In such a situation, the implementing developer may make an uncertified probationary version of the enhanced software available at any time. The probationary version then becomes the default version for any user who requests it by

executing a special command. The remainder of the user community continues to use the previously certified version.

Upon completing the maintenance activity and any unit testing, the developer transfers the software to an independent test environment that is isolated from the environment in which the maintenance was performed. Formal integration testing is performed in this environment and if all tests are passed, the implementation baseline, which includes the test procedures and test results, is compiled therefrom and delivered to the CCB for the Acceptance Review.

If the completed software (implementation baseline) is certified by the CCB, the Configuration Manager is notified that the software components may be readmitted to the secure environment. Extreme care is taken to ensure that only those components that were released for modification are readmitted. At this time, the repaired software is integrated into the LANSCE environment, the previously certified version is superseded by the new version, and the new certified version is made available for general use.

THE ROLE OF SOFTWARE CONFIGURATION MANAGEMENT AT LANSCE

As in the case of more conventionally organized large projects, the comprehensive nature of our software engineering methodology requires that the SCM effort serve as the focal point of all LANSCE software development activities. Therefore, the SCM effort actively participates in the management of all phases of the software project, rather than restricting its scope to mere source code control. The preeminence of the CCB, our primary SCM entity, in controlling the evolution of all LANSCE software projects, is one manifestation of this principle.

The SCM effort is ultimately responsible for the mediation, arbitration, coordination and documentation of all facets of the LANSCE software development effort. To accomplish this task, SCM must be thoroughly integrated with the software development methodology and must provide extensive support for each stage of the life cycle. In the absence of such high-level integration and support, an SCM effort rapidly becomes isolated from the mainstream of the project and tends to focus upon mundane procedural matters rather than the fundamental issues that affect the ultimate quality and overall integrity of the software system.

The LANSCE SCM philosophy postulates two basic principles: modularity and information hiding. The modularity principle asserts that all SCM activities can be classified into either of two general categories: operations that are performed most effectively by human beings, and tasks at which humans are particularly ineffective. Examples of items in the first category include all tasks that benefit from the application of human knowledge and judgement, such as evaluations, assessments, reviews and formulation of policy. Items in the second category are often mundane, tedious and/or repetitive in nature and include such tasks as security and report generation.

The effectiveness of an SCM program in addressing the tasks in the first category ultimately determines the overall success of the program, whereas the effectiveness with which the program deals with items in the second category determines the overheads that are imposed upon the effort. It is the failing of most SCM programs that they address the second category (usually with some sort of automation) to the complete exclusion of the first. The LANSCE philosophy, however, establishes the human-intensive aspects of SCM as preeminent. It then strives to reduce the overheads imposed by the remaining tasks so as to avoid diverting precious human resources from the primary SCM effort.

Within the context of an SCM system, the information hiding principle postulates that software development personnel should require no knowledge of the internal workings of the SCM system. Conversely, configuration management personnel, in particular the Configuration Manager, should require no detailed knowledge of the architecture or internals of the software applications being managed. Clearly, if either of these cases is violated, the effectiveness of the personnel involved is diluted.

Profound problems can develop as a result of violating the information hiding principle and can seriously threaten the viability of the project. Developer morale may degrade drastically as a result of the perception that menial clerical duties are intruding upon the technical environment. Furthermore, considerable resistance may develop within the development community to acquiring and maintaining a level of proficiency adequate to interact with the SCM system.

On the configuration management side of the project, the major danger is that of forcing the Configuration Manager to acquire guru status in order to perform the SCM function. In this scenario, the Configuration Manager becomes an irreplaceable member of the team. Only the Configuration Manager has sufficient expertise to integrate, rebuild or even locate particular software components. The presence of such a fragile single point of failure in any mission-critical system is intolerable.

The LANSCE SCM system has four distinct goals:

- to provide a secure environment for certified software,
- to define a consistent, hierarchical scheme for organizing all software components,
- to aid in the verification, validation, and integration of all software components, and
- to support software managers by documenting the current state of the software project at any instant in time.

The LANSCE SCM system must provide security at two levels: change control and physical security for certified baselines. Change control is the primary responsibility of the LANSCE CCB, as discussed in the previous section. Physical security is ensured by depositing all certified baselines within a data base and severely restricting access thereto. In the LANSCE environment, we are using the Change and Configuration Control (CCC) product from Softool Corporation. Only the Configuration Manager has access to this secure repository, called the Configuration Data Base (CDB).

All software releases from, or admissions to, the CDB must be preauthorized by the CCB and are performed by the Configuration Manager. The mechanics of a release or admission operation are executed by sophisticated automated procedures that require essentially no interaction with developers or configuration management personnel. In this fashion, the preeminent role of the CCB is reasserted, operational overheads are minimized, and the information hiding principle is satisfied. Consequently, the requirements of the classical Configuration Control discipline are satisfied fully and in an optimal fashion.

Providing a hierarchical organization and meaningful labels for all configuration items lies in the realm of the classical discipline of Configuration Identification. Applying the modularity principle, it is clear that formulation of the configuration identification scheme requires application of human judgement and knowledge, but its application to the various configuration items benefits significantly from computer automation. Consequently, we devoted significant resources during the design phase to identifying an appropriate architecture for the CDB. The application of this architecture to the numerous baselines resident within the CDB is performed by automated procedures. At the time each baseline is admitted to the CDB, the procedures appropriately mark and hierarchically organize each accordingly.

Traditionally, the verification, validation, and integration of delivered baselines is under the auspices of the Configuration Auditing function. The support provided by the LANSCE system for software validation and verification is discussed in a previous paper.[7] Integration support is provided by automated facilities that perform software rebuild and rebaseline operations. These facilities provide an extremely low-overhead and highly reliable capability for integrating new or modified software into the LANSCE operational environment.

The management support function has traditionally been the responsibility of the Configuration Status Accounting effort. Because of the care that we exercised in defining the CDB architecture, this aspect of the LANSCE system can be completely automated. A software manager may request reports that document virtually any aspect of the software development project, including module status, work in progress, current revision levels, progress of work, and active baselines.

ARCHITECTURE OF THE AUTOMATED PROCEDURES

The architecture of the LANSCE Software Configuration Management system is designed with security of the certified software as the primary focus. To that end, only the Configuration Manager has access to the configuration data base, the SCM procedures are automated to as great an extent as possible, and only one developer is given responsibility for a deliverable or its components. Using the CCC data base as a single-user system greatly simplifies the maintenance of the CDB and requires less overhead than a multi-user data base. Automation of the SCM procedures contributes to security by eliminating the opportunity for inadvertent errors and reduces the personnel overhead otherwise required by manual procedures. The automated procedures are executed in batch mode, further reducing the human interface. Assigning only one developer responsibility for a deliverable greatly increases the security of the system by eliminating the possibility of multiple efforts colliding during integration testing or, even worse, when the deliverable is admitted into the CDB. For further security, we have isolated the functions of software development, test, and configuration management (as shown in Fig. 1) both to reduce their interactions and to increase the visibility of their contributions.

The limitation of CCC users greatly reduces the risk of accidental corruption and improves the response time of data base activities. In addition, it eliminates the maintenance of protections for various CCC users. The training overhead required to maintain several persons fluent in our SCM procedures and in the CCC product itself is also eliminated. Only the Configuration Manager may start an SCM procedure and then only with the approval of the CCB. Because the procedures operate in batch mode, require minimal input, and automatically produce transaction logs, they require no monitoring by the Configuration Manager.

As an additional security measure, we prohibit software of one deliverable from being released to more than one developer, thereby eliminating the chance of conflicting modifications or of accidently overwriting one set of modifications with another. This simplifies the interfaces needed to support software release and admission functions. In the verification phase of a software admission function, the username of the admitting developer must agree with the username of the release developer. In a software release function, if any modules are already checked out for update the remainder may only be released to the same developer. The deliverable is treated as a unit, increasing the reliability of integration testing and simplifying the tracking of software release and admission operations.

The majority of SCM procedures operate on groups of files that can be organized into lists. For example, in a software release from the CDB, those source files that require modification are copied to the developer's environment, the remaining sources are compiled and object files are made available for linking with the modified routines. This procedure is easily automated by maintaining a list of those files requested by the developer

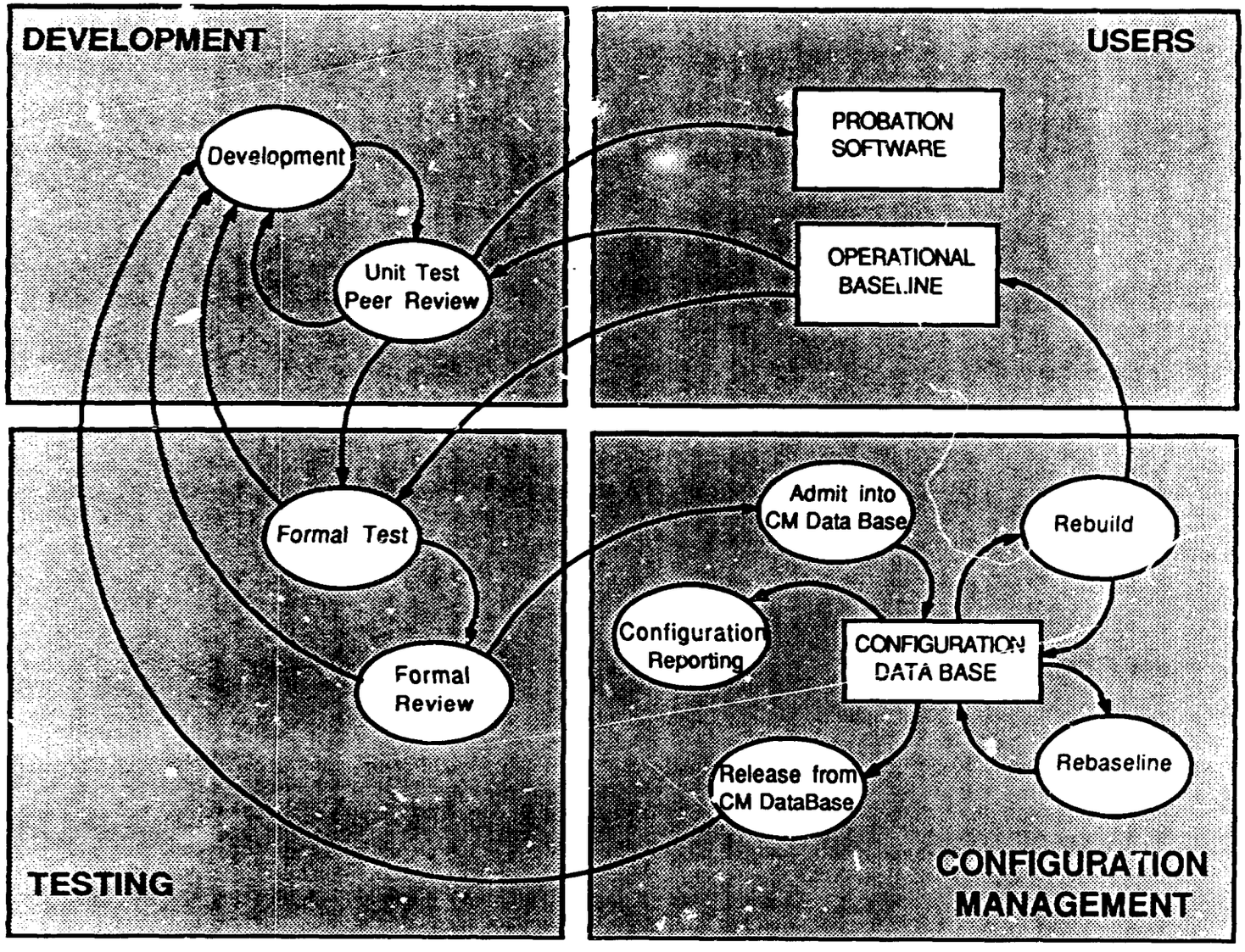

Fig. 1. The LANSCE software system functional relationships.

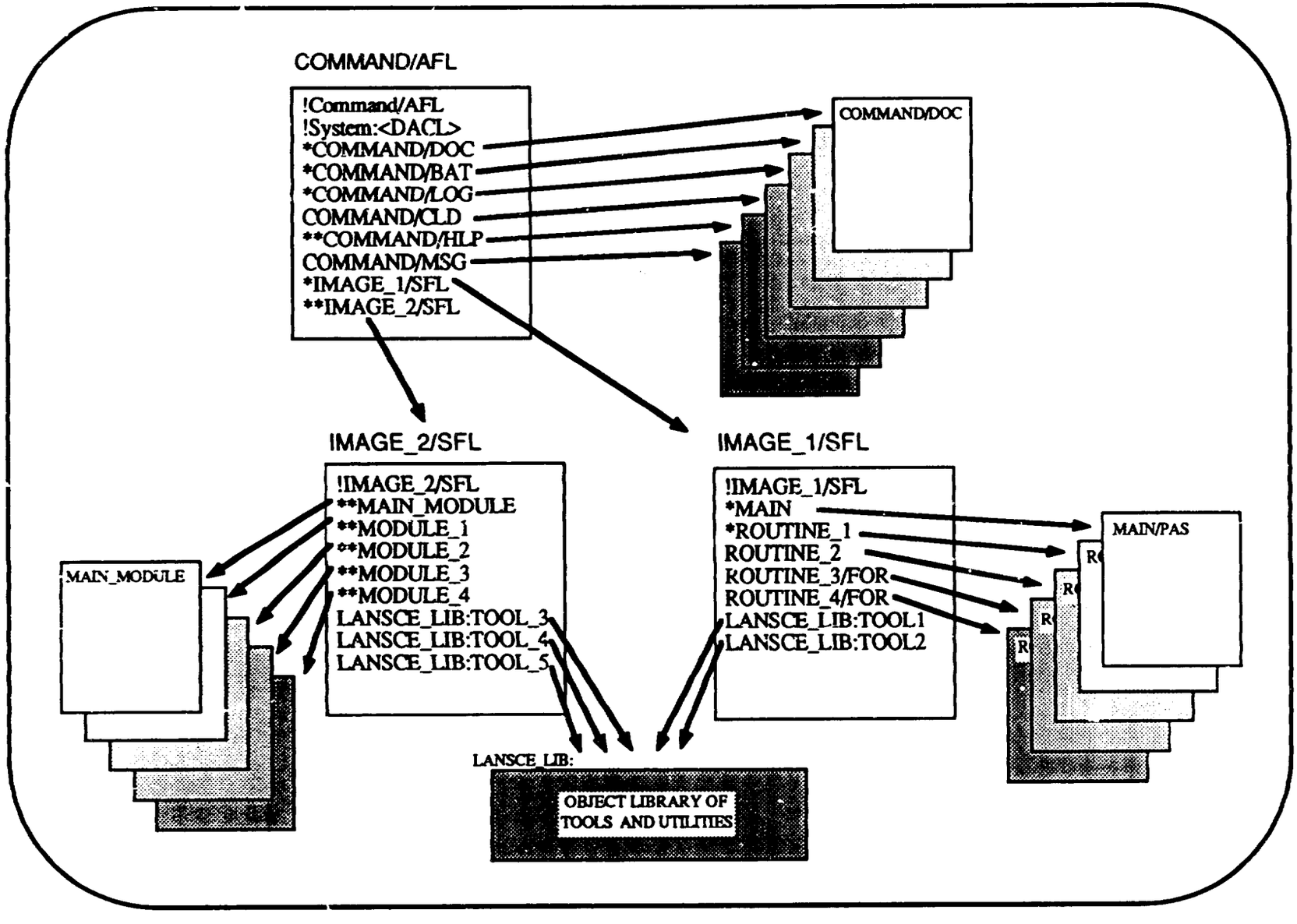

Fig. 2. Relationships between AFLs, SFLs, deliverables, and object libraries.

for modification, and a list of the related files that must be compiled. The use of file lists facilitates the automation of all functions that can best be done by a computer by specifying the entire set of files belonging to each deliverable. With an appropriate organization of the deliverables, the generation of file lists is simplified and virtually maintenance free.

We have adopted the use of file lists to integrate and automate the various functions of theLANSCE Software Section. The file lists are organized by software deliverable and contain only file names, not locations, thereby allowing more general use. Being able to specify the entire set of files by entering only the deliverable name reduces the amount of interaction with the automated procedures, thereby limiting the risk of omitted or incorrectly specified files. These file lists are the key elements that integrate the various disciplines and pave the way to simplified automation.

At LANSCE our file lists support the SCM functions, as well as software development and testing operations. They are used primarily for transmitting files from one environment to the next, and also for the operations within each environment. The Configuration Manager uses the deliverables file lists, greatly reducing the data entry required to perform the various SCM operations. This contributes to lower overhead and increases the reliability of the SCM procedures. The Configuration Manager need only to specify the deliverable name, developer's name, and optional audit information to start an admission or release procedure; the remainder of the process is totally automated. The software developer uses the same file lists for all compilation and linking activities and for submitting the deliverable to the test environment. This integrated use of the file lists directly contributes to reduced overhead and increased security by eliminating manual procedures. Because they are routinely used by the developer for compilation and linking, the file lists are virtually guaranteed to be correct.

For each executable image of a deliverable, we have defined a Source File List (SFL) to list all of the routines that make up that image (Fig. 2). The SFL lists all of the source code files that make up an executable image, and any object library references required by the source code. Also included is the local environment (Include) file, if any. Within the SFL an asterisk preceding a file name indicates that it is certified and resides in the CDB, but is checked out to a developer for modification. Such files are designated "modified." Two asterisks indicate that the file has never been certified and does not reside in the CDB: it is newly developed (new). These indicators are used extensively by all macros and command procedures to determine which files are to be manipulated and where they are located.

The SFL is used in compiling and linking the image, as well as in transferring the source files from development through test, on to the CDB, and back to development. By specifying the SFL name to our compilation tool, either all routines or only the new and modified routines are compiled, depending on the presence of command qualifiers and indicators within the SFL. The SFL name is then given to the linking tool and the object files are automatically linked together. In this way, the software developer maintains the SFL in a very natural way during unit testing, with little or no effort, and it is verified by the linking process with immediate feedback. The SFL is used again after unit testing by the formal test and the SCM functions to compile and link the deliverable in their respective areas.

The Ancillary File List (AFL) lists those files associated with the deliverable that are not listed in the SFLs, including documentation files, test data files, batch command files, and the SFLs themselves. These are designated ancillary files because they are not directly related to the software programs as are the source files. The AFL name is the same as the deliverable name, so by providing the deliverable name as input to an automated procedure, all files associated with a the deliverable are made available to that procedure. The AFL therefore is the primary input to all file transfer functions whenever the deliverable moves among the software development, testing, and CDB environments. In addition, the AFL is used for rebuild and rebaseline operations, thereby providing a concise list of all required filenames. The names within the AFL can be identified as in the SFL with one, two, or no asterisks indicating whether the file is modified, new, or old. As for the SFL, the AFL is maintained by the developer responsible for the deliverable, in a natural way, during development.

The Configuration Data Base

The LANSCE configuration database is designed to facilitate the automation of a majority of the SCM functions and to reduce the amount of data entry required to run the automated procedures. Both of these goals contribute to increased reliability for the entire system. We have selected the Softool CCC database to be the central repository for our certified software because of the security and automation features it supports. The CDB is hierarchically organized, as shown in Fig. 3 and 4. Dependency information is retained in the hierarchy and a configuration naming system is incorporated to satisfy our configuration identification requirements.

The LANSCE SCM system employs only one CCC data base in which all certified software resides. Although this architecture probably results in some degradation of run-time performance, it significantly reduces the overall operational overheads. By retaining all certified information in a single data base, dependencies between the various software systems are simply represented within the data base structures, rather than in the software, as would be the case for a multiple data base approach. Furthermore, the necessity to communicate between separate data bases during routine SCM operations is eliminated completely. As a final bonus, it is not necessary to maintain multiple copies of the CCC macros that perform the various data base automation functions.

The hierarchy of the LANSCE CDB begins at the CCC system (/SYS) level. At this level we define the principal

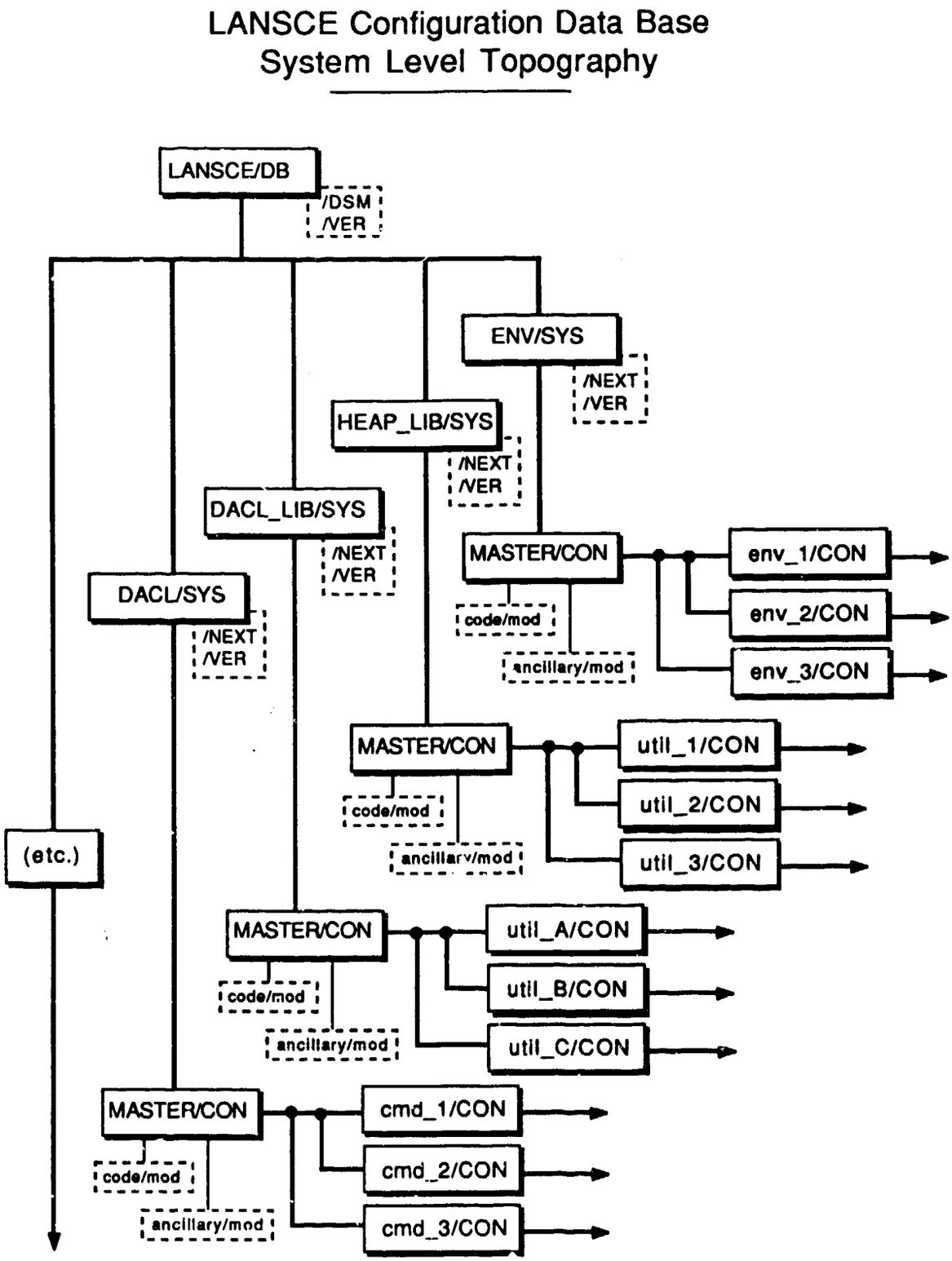

Fig. 3. View of the LANSCE CDB systems and configurations.

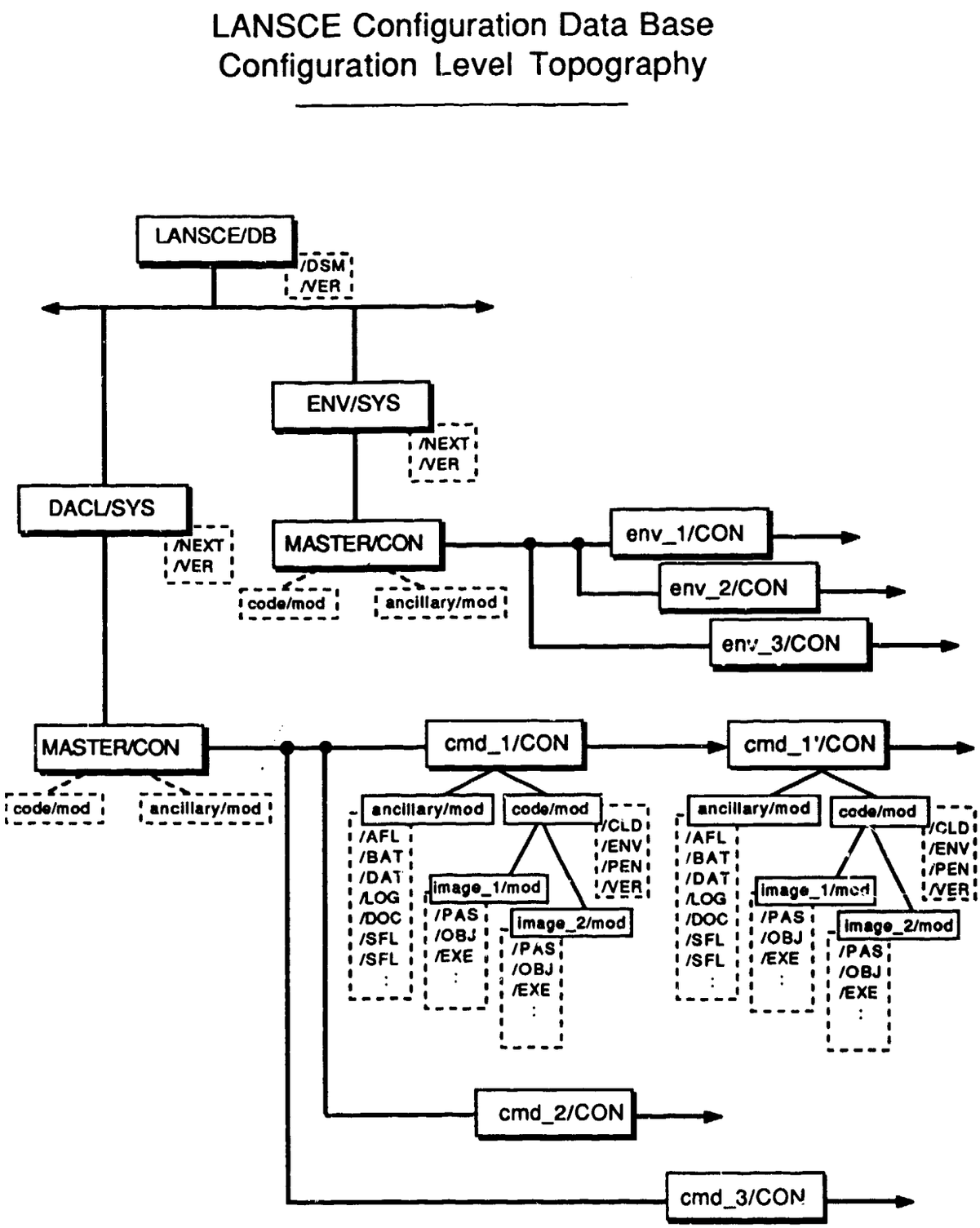

Fig. 4. View of the LANSCE CDB modules and resident texts.

LANSCE software subsystems including environments (global Include files), library-resident utility modules, software tools, and data acquisition programs.

The architecture of the data base automatically specifies the dependency relationships between the various structures within the CDB. At the highest level, each system contains a file called SYSTEM/NEXT that points to the next system in a descending order. This file is used during rebuild and rebaseline operations to indicate the order of precedence of the various software subsystems. For example, the global environments, upon which most of the utilities, tools, and applications software depend, are compiled first in a complete rebuild operation. Next the procedure looks at the SYSTEM/NEXT file in ENV/SYS to determine the next subsystem to be rebuilt. That file points to a utilities subsystem, HEAP_LIB/SYS.

Every certified deliverable resides in a CCC configuration (/CON) data structure of the same name. This allows the automated procedures to easily locate the proper CCC structures for any deliverable, facilitating automation and enhancing the accuracy and reliability of the process. Only the current configuration for each deliverable has the same name as the deliverable; older versions are named with the deliverable name and a suffix. The parent system name of each deliverable is stored in the AFL, directly accessible by all SCM procedures.

The CCC configuration for each deliverable is partitioned into two CCC module (/MOD) data structures, named ANCILLARY/MOD and CODE/MOD, as shown in Fig. 4. The ANCILLARY module contains all of the ancillary files associated with the deliverable. The CODE module contains a subordinate CCC module (/MOD) data structure for each of the SFLs that compose the deliverable. The name of each such subordinate module is identical to that of the corresponding SFL. Each contains the certified source code modules referenced by the SFL, as well as the binary executable image (load module) built therefrom.

Within the configuration for each deliverable, we maintain not only source files, but objects, executables and listings as well. Not all of these files require the same level of maintenance within the CDB. For example, some files such as source code listings and object modules are easily regenerated so only the most recent version need be maintained. Other files require that a history of their evolution be maintained in case recovery of an earlier version is required. The level of maintenance is specified in CCC by the structure traits: ARCHIVE, GLOBAL, and LOCAL. Only the source code and selected diagnostic files are given the ARCHIVE and GLOBAL traits, thereby maintaining a single original version for each such file and an audit trail of differences for each update. Files with the ARCHIVE and GLOBAL traits include the command language definitions, the module sources, environment (Include file) sources, linkage maps, test scripts, test data files, and test log files. The remaining files are defined with the LOCAL and NOARCHIVE traits.

THE LANSCE SCM PROCEDURES

The LANSCE SCM automated tools and procedures support all aspects of classic software configuration management. The structure of the CDB and the design of the applications software directly support the Configuration Identification and Configuration Status Accounting functions by their hierarchical and strictly modular nature. The Configuration Control function is enforced by the software admission and release procedures, which verify the status of the certified software before performing any transfer actions. Our rebaseline and configuration reporting procedures support the Configuration Auditing function. All automated procedures are implemented with modular, general-purpose CCC data structure macros (/DSMs) and host command procedures. These macros incorporate a policy of strict modularity to ensure high reliability and ease of maintenance. Because they are general purpose, they can be reused as needed, thereby further reducing maintenance efforts. Our SCM procedures are highly automated to reduce the opportunity for introducing errors in the CDB and to reduce the overheads inherent in these repetitive tasks.

Each macro and command procedure is documented at the same level as the applications software routines, with a header template describing the purpose, interfaces, global and local data structures, functional description and pseudocode. This documentation is designed to improve communication efficiency among designers and developers, but since the macros and command procedures are executed by a command interpreter, their execution efficiency is degraded. For this reason, we create an execution version for each CCC macro by extracting only the executable CCC statements from the documented version. The documented version is therefore available for maintenance or perusal, but only the extracted version is executed.

The LANSCE macros make extensive use of the CCC HOST Facility for interacting with the VMS environment. We use the HOST Facility for compiling and linking deliverables at admission or release time and for rebuild operations, as well as for sending electronic mail to affected parties at the conclusion of an SCM transaction.

Most of the automated SCM procedures execute in batch mode, thereby operating at a lower priority than interactive users, and freeing the invoking terminal for concurrent interactive processing . All produce a transaction log that is sent by electronic mail to the Configuration Manager and to the responsible developer upon completion of the desired action.

Fundamental to all of the automated SCM procedures are the macros that parse the AFLs and SFLs and the macros that verify the current status of specified files resident in the CDB. The AFL/SFL parsers determine the status of each file in the list by interpreting the identifiers that precede each list entry. The verification macros automatically check that a specified file has the required import or export status. These basic macros are used by all SCM procedures. In addition, the AFL/SFL parsers are employed to streamline the automated software testing.

That these simple macros support such a large number of client procedures and functions is due to our deliberate efforts to design a system that is constructed of small, modular, reuseable building blocks[8].

The software admission procedure is designed to validate automatically each deliverable prior to admission into the CDB. It is list-driven from the AFL and associated SFLs and as such requires only the deliverable name to locate, validate, and transfer all of the files associated with the deliverable. The procedure is batch-oriented so the Configuration Manager can queue multiple admissions and return to other activities while the admissions are processed in the background. The role of the developer in this process is minimal, requiring only that software testing be completed and that the implementation baseline be submitted to the CCB for review.

The Configuration Manager initiates an admission cycle by executing the admission macro and supplying the macro with the name of the deliverable to be admitted and the name of the developer to whom it is currently released. If a configuration does not already exist in the CDB for the deliverable, the Configuration Manager is prompted to verify the deliverable name, thereby trapping spelling and typographical errors. The Configuration Manager is then prompted to enter change reference information that is to be associated with the current admission. Thereafter, the remainder of the admission process proceeds automatically in background mode.

If a prior certified version of the deliverable already exists in the CDB, the admission procedure verifies that the username of the developer associated with the current submission matches the username of the developer to whom the release was made, and all modified files are verified to ensure that they are currently exported for update. The software is then rebuilt in an isolated host directory to ensure that it compiles and links with the certified environments (Include files) and object libraries. If any of these verification steps fails, the submission is rejected and a detailed record of the submission is sent by electronic mail to the Configuration Manager.

After the verification steps, the files identified in the AFL and SFLs as new or modified are automatically imported. The deliverable images are rebuilt and the operational baseline is automatically integrated into the LANSCE environment. Upon completion, the Configuration Manager and responsible developer are automatically notified of the status of the admission by electronic mail.

In the event that a configuration does not already exist in the CDB for the deliverable being submitted, the admission procedure automatically names and inserts a CCC configuration for the deliverable. As part of the process, the subordinate ANCILLARY and CODE module (/MOD) data structures are created, the CODE module is subdivided into subordinate /MODs for each SFL in the submission, and the appropriate traits are set up based upon the expected file types. The admission procedure then continues in batch mode as described above.

In this manner the rote mechanical aspects of the Configuration Identification function are performed automatically. Accuracy and completeness are guaranteed. Furthermore, this feature simplifies the integration of the SCM system into a pre-existing project. Rather than being forced to manually configure the CDB for preexisting certified software, the CDB is automatically configured as the software is loaded into it. This feature significantly reduces the personnel overhead incurred in implementing the SCM system for an existing project such as ours.

The software release procedure parallels the software admission process in that there are verifications, actions, and documentation of the actions. To request CDB-approved files, a developer need only identify with asterisks the appropriate entries in the corresponding AFL and SFL(s). This eliminates the need for tedious, error-prone data entry tasks to identify the required modules and enhances the reliability of the process. The developer then sends an electronic mail message (to the Configuration Manager) that indicates where the marked AFL and SFL(s) are located. The same procedure is used by the developer whether it is the first check-out of software for the deliverable or a subsequent release. The Configuration Manager starts the release procedure and provides it with the developer's name and the full specification of the marked AFL. At this juncture, the Configuration Manager may enter or update preliminary change information that is stored in the CDB until the software is readmitted.

The release procedure verifies the username of the developer requesting the files as being the same as that of the responsible developer, if any components of the deliverable are already released. The files whose names are marked in the AFL and SFL(s) are exported to the developer's directory. Unmarked files are exported to a holding directory and compiled using the HOST Facility. The resulting certified object modules are then made available to the developer for unit testing. These actions are documented in a transaction log that is sent to the Configuration Manager and developer *via* electronic mail, at the conclusion of the release procedure.

Our software rebaseline facility permits the Configuration Manager, with the approval of the CCB, to control the level of detail maintained within the CDB for an individual deliverable (or component thereof), a software subsystem, or the entire LANSCE software system. Depending upon the level in the CDB from which it is invoked, it automatically combines all versions of all archived files into a single version for each. If a configuration (/CON) data structure is within the scope of the rebaseline, a new configuration is created to hold the new baseline.

These features permit the CCB and Configuration Manager to define a series of major and minor releases of the LANSCE system and subsystems and to control the level of detail maintained at each level. This has the effect of logically partitioning the CDB even further and vastly simplifies the management of the project.

This approach benefits significantly from the consistent, hierarchical organization of the CDB. For example, a major release may be associated with each software subsystem (/SYS). Minor releases may be defined for individual deliverables (/CON) and update releases for individual components within a particular deliverable. As such, update releases may be tracked strictly through the file archiving mechanism and change information. Each minor release could correspond to a new daughter configuration for the deliverable, and so forth.

Under this approach, when a new minor release is planned, a new daughter configuration is inserted for the corresponding deliverable, and previous update releases are combined to form a single unified baseline from which to build the new release. Consequently, update releases are only maintained for the most recent minor release.

The rebaseline facility provides the automated support for operations such as these. It allows the CCB and Configuration Manager to decide what is the appropriate level of detail to maintain at each level of the CDB. As such, useless or obsolete information may be eliminated, thereby streamlining the operation of the SCM system, yet an appropriate series of time-dependent baselines is still maintained.

A rebaseline operation may be initiated at any level in the CDB hierarchy, so that all structures at and below that level are rebaselined. The rebaseline procedure appropriately updates the version number files at the Database, System, and Configuration levels. The rebaseline procedure initiates a dependency-mediated software rebuild beginning at the level at which the rebaseline started. Finally, as in all automated SCM procedures, a transaction log is sent to the Configuration Manager via electronic mail detailing the actions that were taken.

The software rebuild facility is provided to support, in a highly automated fashion, dependency-mediated rebuild and reintegration of all or part of the LANSCE software system. All dependency-mediated execution control systems, such as CCC BUILD, UNIX make, and DEC CMS/MMS, require as input, files of dependencies and associated actions to be executed when the dependencies are violated. These files must be provided by some external entity, generally a person, and must perform correctly in ever-changing environments.

Unfortunately, except in the most trivial cases, the environments are exceedingly too complex, and far too dynamic for a human being to be able to identify the dependencies effectively. Consequently, manually maintained dependency files are generally fraught with errors, omissions, and inconsistencies. In the best case, these discrepancies result in overt errors when the dependency file is executed. In the worst case, they cause insidious, hidden problems that may corrupt the software system and remain undiscovered for extended periods of time. Dealing with such problems, when they finally surface, can be an expensive, time-consuming process that seriously threatens the viability of the project.

The LANSCE rebuild facility is unique in its approach to the creation of dependency files. It utilizes the software AFLs and SFLs, as well as information embedded in the hierarchical structure of the CDB, to generate automatically an appropriate dependency file. This takes the guesswork out of dependency generation and guarantees accuracy and correctness. Consequently we can use the system confidently, assured of the fact that we are not jeopardizing the integrity of our project with a tool that is intended to streamline its operation.

The software rebuild procedure is highly automated and dependency-mediated. It is hierarchical and can be initiated beginning anywhere in the CDB. It operates in either of two modes: generation of a rebuild map, thereby compiling a list of required rebuild operations; or generation of the rebuild map followed automatically by its execution. When the rebuild operation is performed, compiled environments, object libraries, and images are recreated based upon their inter-dependencies, and the operational system is updated accordingly. The rebuild map specifies, in sequence, all required rebuild operations. It may be executed automatically by our rebuild facility, as specified above, or it may be retained and its execution initiated manually at a later time. In the automatic mode, all actions are documented and a transaction log is sent to the Configuration Manager at the conclusion of the rebuild cycle.

For reporting the state of the application software, the software configuration reporting procedure lists the current status of all or selected structures, based on the CDB level from which it is invoked. This procedure can list those structures that are currently checked out for modification, checked out by a particular software developer, and/or those structures with a given change reference, or it can provide a summary of all versions of structures selected by level or name. This procedure also creates the above reports in brief or expanded format, where the expanded format includes the change reference and description for each reported structure.

Results and Conclusions

The LANSCE software configuration management system is designed and optimized for our local environment, but is generally applicable to any project for which thorough and comprehensive SCM is a requirement. It provides extensive automated support that virtually eliminates the tedious, time-consuming, and error-prone operations that characterize manual or semi-automated SCM methodologies. Concurrently, it allows project personnel to maintain the integrity of their software systems at a much higher level than is otherwise possible.

Regardless of the functionality inherent in our system, it must be recognized that no automated software configuration management environment can stand completely on its own. At best, such tools only supplement the primary SCM functions that, sadly (or

happily!) must still be performed by people. To succeed at software configuration management, automation is important, but the primary commitment must still be made by human beings. To fail in accepting this responsibility is to sacrifice your project to expediency.

The LANSCE software configuration management system was designed to relieve a small software project of the daunting burdens that accrue from implementing a comprehensive program of software configuration management. We have been successful in meeting this goal. On further reflection, however, it seems clear that its application can have a positive effect on any project, regardless of size. By eliminating the vast majority of rote, tedious and brutally boring operations that otherwise require the attention of project participants, the system not only improves our efficiency, but actually exerts a humanizing influence on the entire process. And after all, isn't that what the Computer Revolution is all about?

References

1. R. O. Nelson, G. Cort, J. A. Goldstone, D. E. McMillan, J. Ross, J. Seal, and D. R. Machen, "The Los Alamos Neutron Scattering Center Data Acquisition System," IEEE Trans. Nucl. Sci., NS-34, 1017, (1987).

2. E. H. Bersoff, V. D. Henderson, and S. G. Siegel, Software Configuration Management - An Investment in Product Integrity. Prentice-Hall, Englewood Cliffs, N. J., 1980.

3. G. Cort, "The Los Alamos Hybrid Environment: An Integrated Development/Configuration Management System," in Conference on Software Tools, J. Manning, ed.,IEEE Computer Society Press, Washington, D. C., 1985, pp. 10-17.

4. G. Cort, J. A. Goldstone, R. O. Nelson, R. V. Poore, L. Miller, and D. M. Barrus, "A Development Methodology for Scientific Software," IEEE Trans. Nucl. Sci., NS-32, 1439, (1985).

5. G. Cort and R. O. Nelson, "The Los Alamos Tool-Oriented Software Development System," in Proceedings of the Digital Equipment Computer Users' Society, December, 1985, pp. 395-399.

6. G. Cort and R. O. Nelson, "The Los Alamos Software Development Tools," in Proceedings of the Digital Equipment Computer Users' Society, December, 1985, pp. 391-394.

7. G. Cort and R. O. Nelson, "A Modular, Automated Software Testing Environment," in Proceedings of the Softool Users' Group, April, 1986.

8. G. Cort, "Improving Productivity with a CCC Tool Kit," in Proceedings of the Softool Users' Group, March, 1987.

Acknowledgement

This work was performed under the auspices of the U. S. Department of Energy, Los Alamos National Laboratory.