♦ Software Configuration Management for the 21st Century

Anil K. Midha

The increasing complexity of both software systems and the environments in which they are produced is pressuring projects to improve the development process using innovative methods. The role of software configuration management (SCM) systems, policies, and procedures that help control and manage software development environments is being stretched beyond the conceptual boundaries it has had for the last decade. One of the key enablers of producing higher quality software is a better software development process. The SCM system must instantiate a quality process, allow tracking and monitoring of process metrics, and provide mechanisms for tailoring and continual improvement of the software development process. More than a dozen SCM systems are now available, each one having a distinct architecture and set of core functionalities. Currently, no single system provides all the key SCM functions in the best form. Thus, a project must assess its real needs and choose the right SCM system to meet its software development challenges. This paper focuses on the characteristics of SCM systems, the SCM challenges for Lucent Technologies, the principal SCM systems being used within the company, and the issues of choosing and successfully implementing the best SCM systems.

Introduction

Software has become an essential component of wide-ranging products. From the microkernel in set-top boxes to various layers of software in telecommunications systems, software is the key differentiator for high-technology products. Technological advances continue to lead the way for producing increasingly sophisticated software products, and their creation requires an equally sophisticated development environment. The increased sophistication of software development environments requires projects to evaluate their development process, tools, and technology.

Software configuration management (SCM) is at the core of the software development environment. The need to use SCM for large software systems has been well understood for about the last 20 years. Increased complexity of products and their development environments not only has sharpened the need for SCM in every software development project but also has expanded its scope. SCM is an important criterion in several recognized standards like the ISO-9000 series and the Software Capability Maturity Model. A better development process—well integrated with SCM—is a key enabler in producing higher quality software.

To meet the future challenges of software development, a project should assess its SCM needs critically, choose an appropriate SCM system that has the essential functionality, and support a well-defined software development process. This paper discusses the characteristics of SCM systems, the future SCM challenges for Lucent Technologies, the SCM systems now being used within the company, and the issues surrounding successful deployment of an SCM system.

The Essence of SCM

Software development is a process within which

Panel 1. Abbreviations, Acronyms, and Terms

ECMS—Extended Change Management System
GUI—graphical user interface
LAN—local area network
MRG—modification request in generic
MR—modification request
NFS—Network File System
RCS—Revision Control System
SBCS—Source and Binary Control System
SCCS—Source Code Control System
SCM—software configuration management
SQL—Structured Query Language
TCP/IP—Transmission Control Protocol/Internet Protocol
VE—Version Editor
WAN—wide area network

specialists trained in certain disciplines (for example, system engineering, programming, and system testing) must work together effectively. In every discipline, people use the appropriate processes to control and manage their activities. For successful software development, however, the project team must follow a well-defined, systematic, and integrated process that facilitates the coordination and management of activities across all disciplines. SCM is the essential thread that joins various software development activities in the form of a disciplined approach. Because SCM covers a wide range of software development activities, people in disparate roles often view it differently. In a broad sense, we can define SCM as a set of tools and procedures that addresses the entire software development life cycle, identifying, managing, and controlling software and related components as they change over time.[1]

SCM provides the discipline to facilitate the continued evolution of a product. The journey of product evolution requires controlling and managing constant changes to internal intermediate deliverables, as well as to the components that constitute products for external use. As opportunities for changes are identified, they must be investigated for their short- and long-term impact on the overall product direction in various development streams—for example, in product releases. All product changes must be coordinated, tracked, planned, implemented, tested, delivered, and maintained over a product's life cycle. A logical change to a product may result in several physical changes to product components. Interdependencies of various logical and physical changes and the propagation of certain changes from one development stream to another add a new dimension to the coordination of changes in various development streams. Furthermore, actual changes may be implemented by a team of project members who must communicate and collaborate to implement all the desired changes successfully.

SCM helps to coordinate all these activities, providing a more effective interface with many other development-related activities like project and test management and requirements traceability.

SCM System Characteristics

A successful SCM system streamlines the entire process of managing product changes, tracking versions and the configuration of components, and coordinating the work of team members, as well as building and releasing product deliverables. The system should be sufficiently flexible to meet the varying and growing needs of projects in different development phases. Thus, an SCM system must provide the following key capabilities:

- *Version control*. The core functionality that every SCM system must have, version control provides the mechanism to keep track of the history of changes to the product components as they evolve over time throughout the software development life cycle. It facilitates the creation of product development streams for maintaining different versions of the product. Any version from a stream can be accessed. Some systems also provide a mechanism by which to identify and merge changes from one development stream to another.
- *Change management*. A process of identifying, evaluating, tracking, managing, implementing, and reporting the requests for changes to a product, change management defines the overall use of the SCM system. The change management process has two aspects: problem tracking and change control. Problem tracking

is the process of recording, tracking, and reporting problems and enhancement requests submitted by end users of a product. Some of these reported issues, problems, and enhancements result in requests for product changes. Change control activity is the process of identifying, tracking, controlling, and managing actual changes to product components throughout the software development life cycle.

- *Configuration control*. A process of identifying, defining, and selecting a collection of components of a product, configuration control allows the use of a criterion for specifying a version of a set of product components.
- *Build management*. A process of efficiently building the whole or a subset of a version of a product from the selected configuration of product components, build management facilitates recordkeeping of a built version, the build environment, and the versions of the selected components in a configuration. Build support is essential in various software development life-cycle phases like unit, integration, and system testing.
- *Process management*. Known as the culmination of all the process support that enables a project to carry out the software development life-cycle activities in a repeatable, streamlined, and (as much as possible) automated fashion, process management has a very wide scope in SCM. It encompasses the process and gates of both the change control and test/verification activities, generation of product- and process-related quality metrics, as well as subdivision and scheduling of development activities.

Other Desirable Characteristics

The SCM system also should provide the following capabilities:

- *Ease of use*. As with any other system, ease of use, which has many dimensions, is particularly important for an SCM system. Of course, the most significant aspect is the simplicity and effortlessness with which members of the software development community can embrace the system's functionality as part of doing their work. Next in significance are the following four aspects: an interface with which users are comfortable or one that they enjoy using (for example, a graphical user interface [GUI] or a command-line interface); the platform on which users normally complete their development activities (for example, a UNIX* environment on a workstation or a version of Microsoft* Windows* on a PC); the integration of SCM functionality with the other software development environment tools (for example, FrameMaker* and MS* Word for document preparation); and the simplicity with which users can identify, select, and extract information for performing their normal development tasks more efficiently.
- *Ease of administration*. An SCM system is unlike many other development tools in that the administrative activities do not end after installation. In fact, installation is just the first step among many to use an SCM system effectively in an organization. Generally, systems require some customization and fine tuning on a regular basis, and administrators must define, set, and monitor project policies and procedures governed by the SCM system. Administrators must also audit data repositories and take timely corrective actions for smooth functioning of the system. The amount of resources (machine and human) for the proper and regular administration of different systems can vary significantly.
- *Support for distributed development*. SCM support for distributed development has increasingly become an essential attribute. Different vendors support different flavors of distributed development. For some organizations, the ability to use an SCM system over a local area network (LAN) is sufficient. Other organizations stress the importance of the system's ability to accommodate development teams at remote locations in using a single centralized system over a wide area network (WAN) for

common development efforts. Still others need distributed databases at each development site at which local development can take place and be synchronized periodically with different remote sites.

- *Integration of SCM functionality*. In some SCM systems, part of the functionality is not provided by the system. Rather, it is supplemented by other associated tools. For effective implementation of SCM methodology, most of the critical functionalities of the SCM system must be well integrated with each other.

SCM Challenges in Lucent Technologies

Typically, Lucent's ongoing software development projects are fairly large, ranging from tens to hundreds of team members working on various components of a product to be integrated. As the company grows, its development activities are becoming geographically dispersed. The requirements to support multiple flavors of products working on a diverse set of heterogeneous platforms for various national and international customers are constantly increasing. In addition, the competitive pressures to produce higher quality products in shorter timeframes continue. These market trends and technological advances, including Internet-based software development, pose some interesting challenges in the SCM area.

Distributed Development

As mentioned above, Lucent project teams are now often dispersed across several geographic locations. For all the team members to perform software development activities effectively, they must all share the same SCM system and follow consistent SCM practices. The chosen SCM system must provide various types of distributed development facilities to meet the requirements of different team configurations. Currently, many projects use a LAN or WAN to access a central SCM repository. The future challenge will be to distribute the SCM repository to the various project sites. Team members at each development site primarily would use a local SCM repository for most of their work. In doing so, the various sites would need to be kept synchronized, and the different software components from each location would be integrated at one of the sites. In such an environment, the SCM system challenge will be to provide facilities for the integration, building, maintenance, tracking, and delivery of the combined product with the same ease and consistent process as provided for the individual components at each site.

Parallel and Concurrent Development

Parallel streams of software products will be developed, maintained, and coordinated to satisfy short- and long-term market needs for a diverse set of customers. From the SCM standpoint, the system must provide the facilities to create multiple streams easily. Developers should be able to implement the product features into a stream, and the SCM system should help migrate the software changes for specific features to the other development streams automatically and transparently. Furthermore, the developers should be able to resolve code conflicts painlessly.

To hasten the development process, more than one developer must sometimes work on the same file at the same time rather than waiting for one developer to finish before another can begin. For this reason, the SCM system should allow multiple concurrent unreserved checkouts of a file by different developers for the same development stream. When a developer attempts to check in a file after the first one, the system should force the merging of any new changes with those previously checked in.

Currently, most projects perform single-thread development—that is, changes are made serially and no merge technology is used. Parallel and concurrent development practices force projects toward more multi-thread development using merge technology, which for some developers is a cultural change. Managing any such change is usually a challenge in itself.

Multi-Platform Support

Increasingly, software development activities are performed on a heterogeneous set of platforms. More and more, products must have an open architecture and must be supported on several different platforms. Often, the target platform for the product is different from the development platform. From the SCM standpoint, this difference requires either that the SCM system work natively on all the development platforms of

a project or at least that its major functionality be available as a client on various development platforms. For effective software development, it is essential that the team members on all the development platforms use the same SCM system and uniform SCM policies.

Integrated Build Management

Building target products from software files is a primary software development activity. In this area, the challenge for an SCM system not only is to provide an efficient build mechanism on every target platform but also to monitor each target product, the environment in which it was built, and the versions of all its constituent software files. If the build management of the SCM system is integrated with both the version control and configuration management functions, all the tracking and manufacturing of the software can be done more efficiently.

Component-Based Development

Reuse at every level is becoming a business necessity to produce high-quality products within increasingly shorter intervals. More and more projects are trying to use industry-standard components, domain-specific reusable software components, or reusable software platforms to build new products more quickly. Each of these strategies requires SCM support from a different angle. The SCM system should provide adequate facilities to search, access, and version control all kinds of reusable assets. It should also provide impact analysis of sharing and/or changing a component in the other configurations in which the component has been used.

Internet-Based Development

Ever-increasing interest in the Internet and Internet-based software development has posed new challenges in the SCM area. First, customers now expect to be connected with vendors via the Internet. They desire the ability to enter product defects and to check the status of fixes. Further, they want product fixes and in some cases even products themselves to be available for downloading through the Internet. This feature requires that the SCM system have Internet hooks in its defect tracking, support, and software delivery and deployment subsystems. Second, it seems that some products will be developed with Internet technology like Java* and ActiveX*. In due course, developers would want the SCM systems to provide a well-integrated development environment for leveraging these new technologies.

These challenges of the software development environment are extending the boundaries of traditional SCM.

SCM Systems in Lucent Technologies

In the past, a few organizations could manage with ad hoc SCM systems developed in house and typically consisting of some scripts that manipulated an underlying version control tool, such as the Source Code Control System (SCCS) or the Revision Control System (RCS). Although such ad hoc systems achieved a modest degree of SCM support with a moderate investment in tool development and maintenance, they rapidly became a resource drain for the organization whenever more substantial SCM support was needed. These systems provided only basic version control and rudimentary configuration management capabilities. As project size, complexity, and the geographic spread grew, the need for a more comprehensive set of SCM functionalities increased as well. Thus, most software development organizations began switching to standard, integrated, and well-supported SCM systems.

Today, the most widely used SCM system within Lucent is the combination of the SABLIME® and *nmake* systems. Both of these noncommercial systems, which were designed and developed by Lucent, are being enhanced and supported by the company's Software Technology Center. For the past ten years, thousands of software professionals at Lucent and elsewhere have been using these products on a wide variety of projects.

Another SCM system that predates the SABLIME system—the Extended Change Management System (ECMS)—is used in a few very large key projects. ECMS was designed and developed by these projects at Lucent for their specific needs, and the company's Platforms and Solutions Software Development Environment supports it.

The next two subsections provide an overview of the functionality and process supported by these

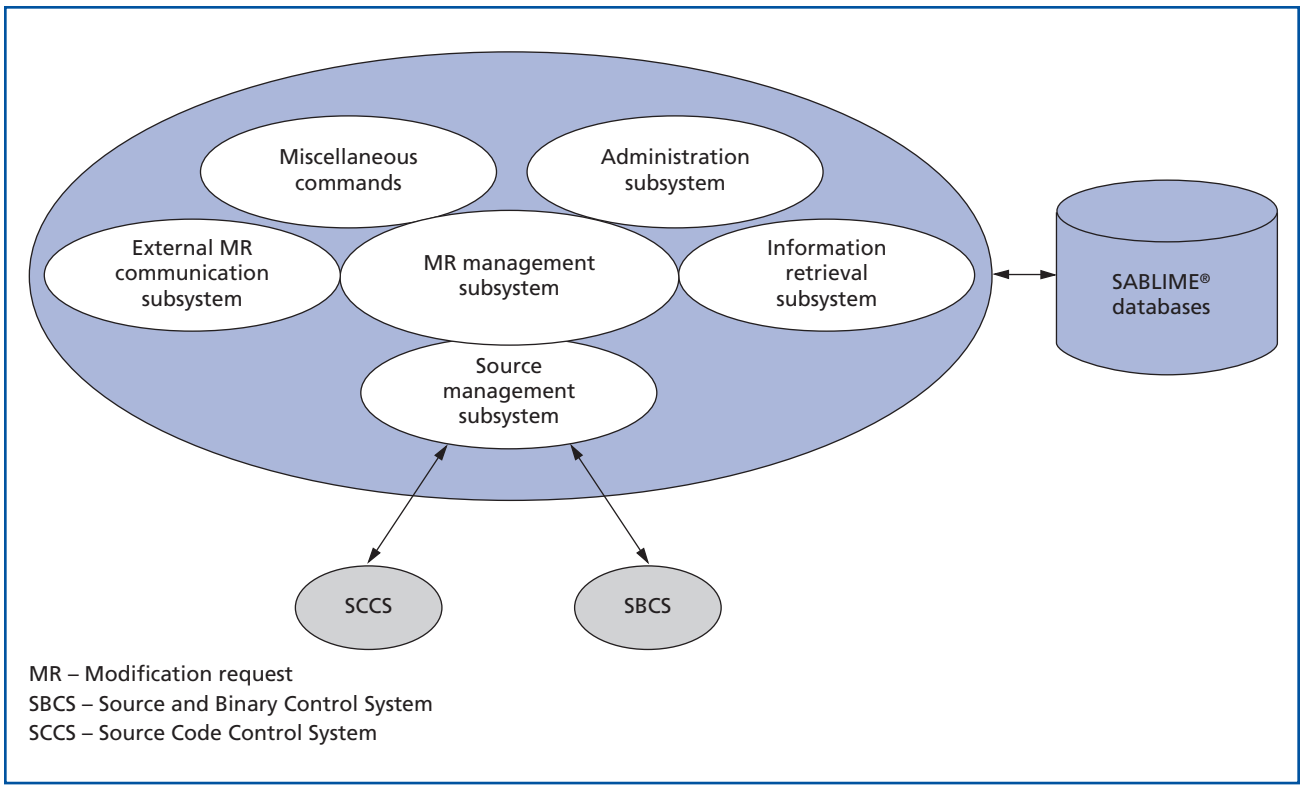

Figure 1.
Overview of the SABLIME system's architecture.

SCM systems, including their relative strengths and weaknesses.

Some projects in Lucent are beginning to use commercial SCM systems. They include the ClearCase CM* system and other related products from Pure Atria, Inc., as well as the Aide-de-Camp/Pro* system from True Software, Inc. These and few other leading SCM systems, like Continuus* from the Continuus Corporation, provide a wide range of SCM functionality. Readers should examine the product information provided by the respective vendors and some independent sources[2,3,4] for the relative strengths and weaknesses of these systems.

SABLIME/*nmake* Systems

Actually two independent products, the SABLIME and *nmake* systems complement each other to provide end-to-end configuration management. The SABLIME system (see **Figure 1**) provides tightly integrated change management, version control, and configuration management capabilities. The *nmake* system provides state-of-the-art product build management.

One of the major strengths of the SABLIME system is that it supports an out-of-the-box process model, which is structured around the change-control life cycle (see **Figure 2**). In the SABLIME system, change requests are tracked both at the product level (modification requests [MRs]) and at each stream of product development (modification requests in generic [MRGs]). Each stream of development corresponds to a planned release of the product, thus supporting concurrent release development.

The change-request life cycle at both levels has enough variability to review requests by one or more project reviewers. It also can take any of the appropriate paths—for example, terminating the request at an early stage, deferring the request for later action, assigning it for a formal study, and/or accepting it for further action for one or more development streams (generally referred to as *generics*). Each change request is assigned to one or more project members with an appropriate priority and due date. If a change request is large or spans several areas of responsibility, it can be subdivided into smaller (child) requests

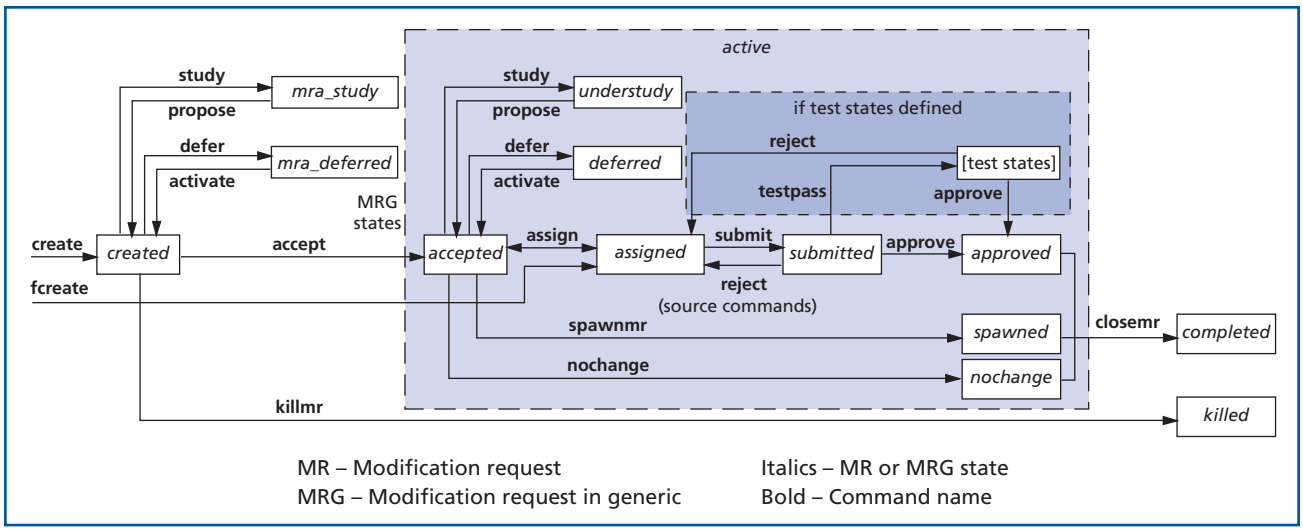

Figure 2.
SABLIME system support for an out-of-the-box process model structured around the change-control life cycle.

and independently assigned, tracked, and managed in different generics.

Using the authorized change request in a generic, the assigned person changes the necessary product components for the generic and completes the work for the change request. Each change request may have to pass various levels of independent tests performed by designated test teams before it can be considered approved for the general release. If a change request fails certain tests, it is rejected back to the project member who originally worked on it for additional work. When a change request is marked approved for all the generics in which it was accepted for work, it can be closed.

The SABLIME system recognizes four different classes of change requests: software, document, firmware, and hardware. It allows customizing the number of test levels, as well as the formation of test teams for each change request class on a per-generic basis depending on project size, rigor, and the desired process. As a change request passes through its various life-cycle stages, the designated project members are notified of the event via e-mail for their subsequent action. This intra-project communication promotes an effective project management process of clearer responsibility, as well as prompt and timely actions. At every stage, the SABLIME system keeps track of who, when, why, and what action was taken, as well as which product components were affected. For each change request, the system requires project members to document detailed information about its description, the solution proposed as a result of the study, the way it is resolved, the rejection notes if it did not pass any tests, and its history.

The SABLIME system generates data to track the quality of the product being developed under it, as well as helping projects track, measure, and improve the quality of their software development process. On a per-change-request basis, the system enables projects to collect and analyze the data to facilitate assessing process quality metrics. For example, a project can collect the data for a release on the development phase in which the faults *have been introduced*, *are being detected*, and *should have been detected*. At the end of the release cycle, the analysis of these data can help discover the gaps between *phase introduced* and *phase detected*, as well as between *phase detected* and *phase should have been detected*. Significant gaps suggest that the project team should improve its process to detect and correct the faults in a development phase soon after the one in which they were introduced. Similarly, various other metrics related to fault type, fault detection, fault prevention, root-cause analysis, and staff effort can help projects identify the lapses in their process. Thus, the necessary actions can be taken to improve the development process and the product's quality.

The SABLIME system provides tightly integrated

version, configuration, and change management. All changes to a product require that the authorized person possess an assigned change request for the desired generic. The SABLIME system tracks all the component changes associated with a change request. It provides two version control branches for each generic: the development branch in which the assigned person makes the changes, and the official branch containing the cumulative result of all the approved changes.

Checking out a component file for changes retrieves its latest version from the development branch and checking in adds the new version at the end of the development branch. The SABLIME system does not allow simultaneous checkout of a file in a generic by two different individuals, thus simplifying the process at the cost of single threading. It allows declaring a component file *common* across multiple generics as long as the development branches for each generic are identical.

For common files, as long as the change request is authorized for all the common generics, the SABLIME system automatically propagates all the changes completed in any of the common generics. Currently, for uncommon files, the developer must make the corresponding changes manually for each generic. A future version of the SABLIME system will allow the automatic propagation of changes for uncommon files. Apart from common files, developers can work in different generics without ever interfering with each other. For each generic, the system allows automatic detection and creation of dependencies between change requests for logically and physically dependent changes in the files.

The SABLIME system provides a choice of maintaining version history under either the SCCS or the Source and Binary Control System (SBCS) version control tools on a per-file basis. SCCS allows versions of only ASCII files, and it has some line-length and special character limitations. SBCS allows versioning of ASCII and non-ASCII files, has no line length or special character limitations, and provides data compaction for both the initial version and subsequent changes. SABLIME keeps all the version-controlled files in a central repository.

The distinctive characteristic of the SABLIME system's configuration management feature is the ability to define and extract a complete or partial configuration of a product's files based on a group of change requests. Such a group can be named and stored in SABLIME and used later for retrieving the corresponding configuration. While retrieving a desired version, SABLIME ensures that the dependent change requests are automatically included in the group. The extracted product configuration can be built using the standard make facility or the *nmake* tool.

The SABLIME system supports a distributed development environment networked over the Network File System (NFS), Transmission Control Protocol/Internet Protocol (TCP/IP), or the Datakit® virtual circuit switch. The repository for the managed product resides on a centralized server that can be accessed by all users simultaneously over the LAN or WAN. Centralized databases provide ease of administration and data auditing. SABLIME's data repository is a simple, efficient flat ASCII file-based relational database. It provides an extensive set of standard detailed, summary, and user-customized reports, as well as raw data that can be post-processed using other report generators. In addition, pie charts, bar graphs, and tabular reports can be obtained. Data can be extracted using powerful Structured Query Language (SQL) queries.

The SABLIME system offers X Window System*- and Microsoft Windows-based GUIs, a cursor-based form interface, and the standard command-line user interface. Each user interface provides extensive data validation, on-line help messages, project-specific menus, and defaults. In addition, screens can be customized for labels, field names, and for their optional or mandatory use. Projects can customize their SABLIME usage by collecting additional project-specific attributes for the change request at both the product and generic levels. SABLIME has a unique feature that allows cooperating projects (either in the same or different location) to exchange information about change requests and their status.

The *nmake* system has many features that clearly make it one of the most sophisticated build engines in the UNIX environment. It provides for an efficient build environment supporting parallel build tasks on a

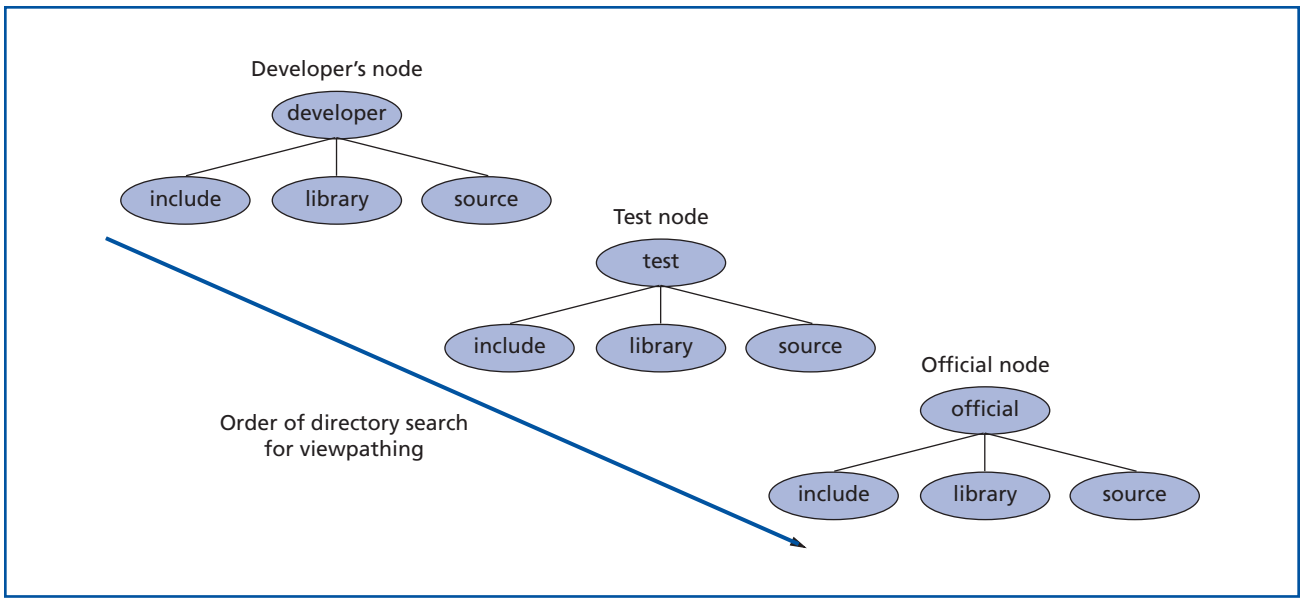

Figure 3.
Viewpathing, which enables project members to share code while maintaining private development areas.

single host, as well as the ability to distribute build tasks to a network of build hosts. In addition, it is efficient in processing makefiles because it works on a precompiled version of the makefiles and preserves the status of every build.

One of the most powerful features of *nmake* is viewpathing, which enables project members to share code while maintaining private development areas. This helps to maintain project isolation and sharing at the desired level while maximizing the use of disk space. Viewpathing also provides an efficient workspace management feature that allows users to build against a common source baseline, override selected files with local versions as desired, and reuse common source and object files without copying them. If the viewpath is set as shown in **Figure 3**, *nmake* first checks the developer's node, then the test node, and finally the official node for required files.

The *nmake* system uses an extensive rule-based language that allows projects to define custom assertion rules and provides dynamic determination of implicit prerequisite dependencies. This facilitates the creation of concise, consistent, and flexible makefiles. The *nmake* system's support for almost any source language (for example, C, C++, M4, and FORTRAN) makes it a versatile build tool for a wide range of application domains.

Extended Change Management System (ECMS)

Another SCM system, ECMS (which was mentioned earlier) is being used by some key, very large software development projects in Lucent. The change management process supported by ECMS is similar to that of the SABLIME system.

The distinguishing characteristic of ECMS is its featuring concept for version control. Essentially, the featuring concept allows developers to apply individual changes in a release (characterized as a set of features) and have them *flow* to other designated releases without having to go through a formal merge process and manually review all the file differences irrelevant to the change. The implementation of featuring is similar to the `#ifdef` construct in C language. Blocks of text in source files can be made conditional on a particular feature by surrounding the text with `#feature` and `#endfeature` constructs. The files using this construction are called *featured files*. When one of these files is checked out for edit, all its conditional text and controlling conditions are present. A specially enhanced editor called Version Editor (VE) is sometimes used by developers to help screen and manage the complexity of many levels (sometimes nested) of the `#feature` constructs.

When a featured file is retrieved for building or testing, ECMS evaluates the conditions according to a

specified feature set, and it puts the resulting unfeatured text in an extracted copy of the file. Thus, a user making a change to a featured file must check in the file before testing it.

By means of featuring, ECMS provides a form of parallel development. Different generics can be defined to use different feature sets. Therefore, ECMS can provide direct support for files that must be similar but not completely identical across a set of generics. The file is made common across the generics, and the differences between the generics are confined to appropriate feature blocks. ECMS also economizes on space by having just one working branch shared by the common generics.

Criteria for Choosing an SCM System

Besides the SABLIME/*nmake* systems and ECMS, projects can choose a commercial SCM system. Both the internal Lucent SCM systems mentioned above and leading commercial systems have enough capabilities and are sufficiently scaleable to work reasonably well in many projects. None of them, however, can be considered a perfect system that provides all critical SCM functions in the best form. Therefore, a project must assess its real SCM needs within its budget—that is, which SCM functionality is most important and which system best meets its requirements while minimizing custom development.

A streamlined and well-understood software development process is crucial to any software project. *The SCM system chosen for a project is the heart of the software development environment. Thus, the system must instantiate the desired software development process.* A well-designed and integrated process management function ensures a better quality software product and improves project productivity by enhancing coordination of individual contributions into a harmonious whole.

The cost of providing an SCM solution has several components and varies significantly. In addition to the license price for the bundled and unbundled SCM functions, two other cost factors must be carefully considered:

- The hardware resources necessary to achieve acceptable performance, and
- The human resources needed to administer and maintain the SCM system.

Furthermore, the cost of producing and maintaining custom developed software in addition to an out-of-the-box system can be significant depending on the ease of use and flexibility of the SCM system.

The SABLIME system provides an out-of-the-box process that is flexible, customizable, and tightly integrated with an adequate configuration management functionality. SABLIME's concepts embodied in its process are simple and easy to understand, and they have a shorter learning curve than comparable products. At this time, SABLIME is an order of magnitude less expensive to license than any other SCM system available. It needs no additional hardware for acceptable performance and requires only minimal administration. However, the system does not support distributed development through which databases could be split at geographically dispersed sites with changes synchronized at specified times. In addition, it does not currently provide parallel development capabilities for merging or migrating changes between generics. This capability, however, is planned for a future version. Although *nmake* complements SABLIME for build management, the two systems are not tightly integrated.

The change management system of ECMS is not as comprehensive or flexible as that of SABLIME. For example, it neither supports the child change request concept nor allows a flexible number of test states. Even though the featuring concept of ECMS is very powerful in supporting change migration across multiple development streams, it clutters the source code files with `#feature` constructs. These constructs make the source code files increasingly difficult to understand and maintain, particularly because the project must support many different releases. ECMS is built on top of SCCS. Thus, it supports only ASCII files. The ECMS system only supports a command-line user interface. Like SABLIME, ECMS does not have an integrated build management system, so *nmake* or any other build engine can be used for building the products.

As the importance of the use of SCM receives increasing recognition, the landscape of available SCM

systems is changing rapidly. Projects must relearn what to expect from an SCM system as the functionalities offered continue to evolve and new products enable new approaches to the existing complex issues of rapid software development. Staying abreast of SCM systems requires considerable ongoing effort. Experimenting to locate and implement a good SCM system is time consuming and costly. Furthermore, difficult-to-achieve consensus and the cooperation of a large team of software developers are required. Settling for a poor system or none at all can be much more costly although the costs may be less obvious.

Conclusion

As software systems are becoming more and more complicated, the development environments needed to build these systems are becoming equally complex. SCM has become the central pillar that supports various facets of software development, particularly the software development process. SCM in its traditional role primarily provided version and configuration control. To meet the software challenges of the next century, SCM systems extend further into the areas of change, process, workspace, build, and release management while supporting a quality process. At present, no particular SCM system provides all these capabilities in the best form at a reasonable price. Projects should examine their key needs critically and choose an SCM system that not only supports but instantiates the desired software development process. A good SCM system is a key to developing high-quality software products.

*Trademarks

Acknowledgments

The author wishes to thank Mark Heimerdinger and John Snively, who provided valuable assistance in the development and review of this paper.

References

1. W. Rigg, C. Burrows, and P. Ingram, *Ovum Evaluates: Configuration Management Tools*, Ovum, London, 1995.
2. S. A. Dart, "Not All Tools Are Created Equal," *Application Development Trends*, Vol. 3, No. 10, Oct. 1996, pp. 39-50.
3. T. Parker, "Software Configuration Management Tools," *UNIX Review*, Vol. 13, No.11, Oct. 1995, pp. 91-96.
4. R. D. Cronk, "Tributaries And Deltas," *Byte*, Jan. 1992, pp. 177-186.

Further Reading

– S. Cichinski and G. S. Fowler, "Product Administration Through SABLE and *nmake*," *AT&T Technical Journal*, Vol. 67, No. 4, July/Aug. 1988, pp. 59-70.
– S. A. Dart, *The Past, Present, and Future of Configuration Management*, Software Engineering Institute Technical Report CMU/SEI-92-TR-8, ESC-TR-92-8, July 1992.
– D. B. Leblang, "The CM Challenge: Configuration Management that Works," *Configuration Management*, edited by W. F. Tichy, John Wiley, New York, 1994.
– M. Cagan and D. Wiborg-Weber, "Task-Based Configuration Management: A New Generation of Configuration Management," *American Programmer*, Vol. 9, No. 10, Oct. 1996, pp. 21-28.
– G. Horton, "Building a Software Development Infrastructure," *American Programmer*, Vol. 9, No. 10, Oct. 1996, pp. 8-11.
– I. Sommerville, Sixth International Workshop on Software Configuration Management, Association of Computing Machinery, Special Interest Group on Software Engineering (*ACM SIGSOFT) – Software Engineering Notes*, Vol. 21, No. 4, July 1996, pp. 54-57.
– G. S. Fowler, J. E. Humelsine, and C. H. Olson, "Tools and Techniques for Building and Testing Software Systems," *AT&T Technical Journal*, Vol. 71, No. 6, Nov./Dec. 1992, pp. 46-61.
– T. R. Hsueh, T. F. Houghton, J. F. Maranzano, and G. P. Pasternack, "Software Production: From Art/Craft to Engineering," *AT&T Technical*

Journal, Vol. 73, No. 1, Jan./Feb. 1994, pp. 59-67.
- D. G. Belanger, E. E. Sumner, Jr., and P. J. Weinberger, "Research in Software," *AT&T Technical Journal*, Vol. 71, No. 6, Nov./Dec. 1992, pp. 62-71.
- D. G. Belanger, G. D. Bergland, and M. Wish, "Some Research Directions for Large-Scale Software Development," *AT&T Technical Journal*, Vol. 67, No. 4, July/Aug. 1988, pp. 77-92.
- C. L. Pettijohn, "Achieving Quality in the Development Process," *AT&T Technical Journal*, Vol. 65, No. 2, Mar./Apr. 1986, pp. 85-93.

(Manuscript approved March 1997)

ANIL K. MIDHA, formerly a distinguished member of technical staff in Bell Labs' Software Technology Center, was the lead architect for the SABLIME configuration management system in the Software Practices and Technology Department when this paper was developed. He was responsible for the SABLIME system's architecture, design, and development, as well as for the training and consulting services on software configuration management tools and processes. Mr. Midha holds a B.E. degree in electronics and communication from the University of Roorkee in India, and an M.Tech. degree from the Indian Institute of Technology in New Delhi. ◆