Evolution of Configuration Management

Rudy Bazelmans

ABSTRACT

The practice of documenting the components of a product, identifying the components of a product, controlling changes to the product and tracking those changes are part of a task called configuration management. This paper discusses the origins and evolution of the term and surveys the software tools which have evolved to support that need.

1. Introduction

Configuration Management (CM) is the organization, documentation and tracking of system configurations. Systems can be thought of as any item which consists of multiple parts. These systems include mechanical systems with sub-assemblies, electrical systems with PC boards and software systems with modules.

The definition of configuration management (for any of the systems types mentioned above) has not changed much in the past two decades. The changes which have occurred have primarily involved their application and the automated support provided. One of the best (and earliest) definitions of configuration management can be found a the DoD Military Standard published in 1968 [MIL 68]. This standard defines CM as a management procedure which includes the following components:

Identification:	The identification of the components which comprise a particular version of a system.
Control:	All changes to the system must be controlled and identified.
Status:	The status of all changes to the system must be recorded and reported.
Audit:	The system must be checked for compliance with the configuration documentation.

My interest in configuration management in the software industry prompted me to collect books and papers on the topic. Of the publications I found, most dealt with CM on the UNIX Operating System. Because the literature is dominated by discussions centered around UNIX and because of UNIX's popularity within academia and industry alike, I will primarily discuss the evolution of CM within that environment and my impressions of that evolution.

2. Origins of CM

The discipline of CM seems to have evolved out of the need to control the production and maintenance of complicated products. The first use of the term CM was in Industrial Management. There are numerous articles on the topic dating back to the mid-60s. [JUA 79] The concept was apparently invented by Henry Ford in the early 1900's in order to organize the production of the first automobile[1]. As products began to use electronics, the use of CM was expanded to include electronics and eventually software. Most of the work in the area of CM seems to have been done by large corporations and government. This is most likely due to the fact they had control over the larger, more complicated systems.

[1]Information on the origin of CM in with Henry Ford came from a manager of an engineering department. Although it makes sense, I have been unable to prove or disprove this statement.

The Department of Defense (DoD) is an organization which is heavily involved in the development of standards. Military Standard (MIL-STD) 480 [MIL 68] was published in 1968 and defines CM. This standard was later revised to include the topic of software and re-issued in 1970 as MIL-STD-483 [MIL 70]. This document prescribes the use of CM techniques in the production of systems for the military. The standard (which was paraphrased in the introduction) defines what CM is but doesn't specify the method which is used to accomplish it.

The IEEE produced a standard for configuration management in 1983 [STD 83] which outlines the structure which should be used to develop a CM plan. The definition of CM which is used within that document is broken down into the same categories as the definition used by the DoD in 1970; namely, configuration identification, control, status and audits.

Hardware and software companies have a wide spectrum of support for CM. Some of the smaller companies have completely ignored the issue and assume the current employees will stay with the company forever. As the companies and the products increase in size, the hardware aspects of the products usually come under CM control. This is done using completely manual systems in some companies to completely automated systems in others. The software aspects of the products also seem to be controlled as a function of the size and complexity of the product and the company. Many software engineers seem to view CM as an unnecessary evil (similar to the way they view documentation) and as a result the quality of the tools seem to be low in most corporations.

3. Evolution of CM Tools Within the Software Field

Over the years, the software industry has viewed the area of CM as a set of independent development controls. This is evidenced by the evolution of software tools to aid CM. The aspects of CM as it relates to software seems to have encompassed the following major topics:

- Source Code Modification Control and Tracking
- System Building
- Bug reporting and Tracking
- Product Version Identification

One early (but well known) tool developed by Rochkind in the early 70's was the source code control system (SCCS) [ROC 75]. This set of programs was first available on the IBM 370 running under OS. By 1975, the programs were available on the UNIX operating system on a DEC PDP-11.

SCCS is a set of programs which allow developers to track the modification of text files (source code, documentation, test cases etc). The system provides the ability to retrieve any version of a source program and allows the developer to produce new versions of programs while supporting the older versions. The key features of the SCCS system are:

Storage:	The use of disc storage was minimized by only recording the modifications made to the systems rather than maintaining separate copies of each version of a file. Studies have found these savings to be significant [ROC 75].
Protection:	Access to files is controlled by SCCS and modifications to particular files (and particular versions of those files) may be restricted to certain individuals.
Identification:	Each file can be stamped with information identifying the author, version and the date and time of modification. This information can be propagated to the object files produced from these files.

Documentation: The system prompts the user for a reason for all modifications and records who made the change, what was changed and when it was changed. This information can later be retrieved and used to report modifications of programs.

By 1977 SCCS had undergone nine major releases (five on the IBM OS and four on UNIX) [GLA 78]. Among the major improvements since the inception of SCCS are the following:

Statistics: The system was used to collect statistics on the production of software. These statistics were used to further improve the existing tools and helped justify new tools.

Propagation: The system automatically propagated bug fixes from previous versions into the newer versions. This caused problems when developers discovered new characteristics in programs that hadn't been modified. This feature was eventually replaced by a system to partially automate the integration of bug fixes.

Bug Reminders: As automatic modification propagation was eliminated, a function was added which sent a message to the developer whenever a module was retrieved which hadn't been upgraded with an existing fix. This characteristic of the system proved annoying and since it was usually prematurely defeated my the users of the system, it was later eliminated.

File Identification: SCCS was enhanced to provide a standard marker in source files to allow a program to automatically retrieve the version identification from files.

Efficiency: The efficiency of the system was improved as the usage increased. Attempts to increase efficiency by using binary (versus readable ASCII) data files proved ineffective and inhibited the manipulation of the data files. The database files were returned to pure readable ASCII.

By 1978, it was recognized (in the UNIX community) that there was a need for a tool which would automate the construction of large systems of programs. Feldman produced a program called Make [FEL 78] which could build systems using a script (or database) called a *makefile*. This file defined the components of a system and show the time ordered dependencies of the components. Make was intelligent enough to only rebuild those components of the system (or the system itself) if the components (or their dependencies) had been modified. This was done by comparing the date and time of last modification for all the components of the systems as defined in the Make script. This saved considerable processing time but more importantly, it assured that the system was being correctly built and up to date.

At about the same time as the Make program was being developed, the authors of SCCS realized a need for many individuals to work in concert in the development and maintenance of software systems [GUY 80]. To solve this problem, a higher level interface program was developed to handle the use of a common set of programs by a group of developers. This program (called inter) may have been the predecessor of a much larger system developed by Bell Labs called *MESA* which is described below.

Several years after the development of Make, a need was seen for improved cohesion between the system building capabilities of Make and the source control capabilities of SCCS. An augmented version of Make was developed to support this need [BRA ??]. This new version of Make could automatically retrieve files from SCCS control as required to build a system. The new version also had more innate knowledge so that users didn't have to know as much about Make programming.

3.1. Change Management

Beginning in 1978, a concern over bug tracking and change tracking seemed to develop. In 1979 a Change Management Tracking System (CMTS) was developed which provided the following features [BOE 79]:

- A Modification Request (MR) tracking system
- A Personal (modification) Tracking system
- A flexible report generator for both systems
- A set of programs to manage and use the system

This system was built to be easy to use and customize. It even supported use by multiple projects on the same machine. This system is still being used today at Bell Labs.

3.2. The Revision Control System

Walter Tichy at Purdue University produced a new source control system called the Revision Control System (RCS). This project was intended to improve some of the deficiencies of SCCS as it was being released by AT&T with UNIX. Among the improvements made are the following [TIC 82]:

User Interface:	Users could now specify their file names when manipulating files rather than referring to the SCCS database file names. The commands were also more mnemonic.
Named Versions:	Program versions can be named instead of being restricted to numbers as in SCCS.
Performance:	By using a different algorithm for storing modifications of files, access time for the more recent versions of files was significantly reduced. Infrequently used features of the system had also been eliminated.
Flexibility:	The system was more flexible in it's ability to support the group production of software.

3.3. Applying SCCS to a Large Project

During 1977, Bell laboratories was beginning to work on the Electronic Switching System (ESS). This system would be considered large by anyone's standards; 100,000-100,000,000 lines of code, 100-1,000 developers, 1,000-10,000 modules. The system had several complex components and a number of major subsystems. Yourdon claimed at the time that the task was "nearly impossible" [BAU ??]. Besides being huge, the task would undergo incremental development and continuous support.

It was decided that SCCS would not be appropriate for the project because it assumes that there is one view of the product, namely that all changes which are made today will be included in tomorrow's release. Instead, a system called the Change Control System (CCS) was developed. This system had all the capabilities of SCCS but also supported:

Multiple Views:	The system could support multiple views of the same product. This allowed multiple versions of the same product to be going on simultaneously without interference.
Object Code:	Because of the complexities of the project and the amount of parallel development, it was impractical to require that object code be reproduced continuously. CCS had an algorithm for computing the minimum difference between object modules and remembering those differences.
Change Model:	The system used a finite state machine (FSM) to help track the numerous views of the system. This FSM was capable of providing status on the progress of changes as well as other information concerning the project.

ESS underwent over 4,000 modifications during it's first year of operation and CCS was considered a success. This system is one of the propriety tools used by Bell Labs.

3.4. Other Events of the Era

At about the time that MAKE, SCCS and RCS were being used with UNIX, DEC developed a set of programs with similar capabilities (CMS/MMS). MMS is similar to Make and added nothing significant to the technology. CMS is an enhanced version of SCCS. One of the nicest features was that version numbering and retrieval of files was part of the operating system. The fact that the manufacturer of the most popular UNIX hardware (DEC) had recognized the need for these tools in their VAX VMS operating system helped bless their use [DEC 82].

3.5. Bell Labs "Standard" CM Tools

Although the CCS system was used on the ESS project at Bell Labs, little has been published concerning it's use. Internal training at Bell Labs promotes the use of two other tools. These tools provide higher level interfaces to the SCCS and Make programs. Both tools were developed prior to 1982.

Instead of SCCS, users interact with a program called the Management Environment for Software Administration (MESA). As discussed previously, SCCS (and similarly RCS) have difficulty managing projects with parallel subsystem development and continuous enhancement. MESA provides the following features over SCCS:

Hierarchy Support:
MESA supports a three level hierarchy for projects: Project, Subsystem and Book. This hierarchy is an immense help in the production of large projects.

User Interface: The commands have been cleaned up. For example, the user can now specify the name of the file which he wants to retrieve from the source management system rather than specifying the name used by the source management system (as is done in SCCS but not in RCS).

SCCS Isolation: The user is now isolated from the SCCS (MESA) data files because they are placed in a separate directory system called the MESA/SCCS Pool.

Another program which is in common use at Bell Labs is the Object Generation System (OGS). In essence, OGS is to Make what MESA is to SCCS. OGS is a layer of software between the user and Make which provides the following capabilities:

User Interface: The user is isolated from the low level details of makefiles (scripts used by Make) because generic Makefiles are provided. These files are also stored in a common directory.

Multiple Targets: The user can easily produce products targeted for different machines. The files are stored in the same workspace using a directory hierarchy.

Workspaces: It is possible for a user to do development with a workspace which is only partially populated with files. This avoids the clutter, disc space and danger of having all the source files for a system within the current directory structure.

The current development methodology used at Bell Labs involves the use of MESA, OGS, CMTS and ATK. The Administrator's Tool Kit (ATK) is a set of programs used to aid in the use of the other CM tools. For example, ATK can set up a filesystem hierarchy to mirror another in preparation for the use of MESA. These tools are proprietary and are not available outside of Bell Labs.

3.6. Other Interesting CM Tools

One interesting tool which is available from Human Computing Resources Corporation is called CoCo (Configuration Control). This tool provides a means to manage change requests. A user submits a standard change request to the CoCo program. CoCo sends the request to all project members and solicits comments (using the UNIX mail program). The comments are collected and mailed to a change review board who decides on the change. The decision is then distributed to the project members and logged in a history database. This tool can be used in combination with all the other tools mentioned so far.

There are two other tools available which offer some advantages over SCCS and RCS. The Change and Configuration Control (CCC) system from Softool Corporation provides a partially integrated environment which controls object code as well source code, allows named versions, supports encryption and compression, on-line help and complete error recovery [CCC 84]. The Aide-De-Camp (ADC) system from Software Maintenance and Development Systems Inc. provides an integrated environment which includes named versions, support for families of products, correlation of changes involving multiple files, on-line help and reporting capabilities [ADC 84].

3.7. Yet Another Bell Labs CM Tool

There is yet another CM tool which is used within Bell Labs [ERI 84]. This tool (called Build) is used to generate software and extend the capabilities of the Make program. It supports the ability to maintain separate views of a product in order to develop various versions of the system simultaneously. Basically the procedure used to develop the software is as follows:

1. The developer builds a directory structure under his own directory which matches the directory structure used to build the program.
2. This directory structure is populated with only the files being changed.
3. The shell environment variable VPATH is set to the root of the directory which holds the master software.
4. When the developer BUILDs the system, all unresolved file references in the makefile are resolved by linking the file from the directory structure specified in the VPATH variable into the correct location the user's file structure.
5. After the new version of the system is built, the files which were linked into the current filesystem are removed.

Using the Build program allows users to test their changes independently of others working on the same set of programs. It also avoids clutter in the directory and saves directory space over the conventional Make methods.

Another benefit offered by the Build program is that it assures the developer that the makefile is correct since any file which is not correctly specified within the makefile will not be retrieved from the filesystem specified by the VPATH variable. This configuration audit function is not provided through the OGS program.

4. State of the Art

The most powerful CM tool which I have seen is the Domain Software Engineering Environment (DSEE) developed by Apollo [LEB 84]. DSEE provides an integrated set of tools which support the functions of SCCS, Make and more. DSEE consists of several major components called managers. There is a History Manager, a Configuration Manager, a Task Manager, an Advice Manager and a Monitor Manager.

4.1. History Manager

The History Manager provides source code control similar to SCCS and RCS. The history manager also provides numerous other capabilities. It provides an interactive, multi-window merge capability to aid in the merging of code changes. It supports the use of conditional compilation by passing flags to the compiler based on the system configuration being developed. The history manager also compresses leading blanks from source files; this saves approximately 20% of disc space utilization.

4.2. Configuration Manager

The Configuration Manager uses a *system model* which contains the dependencies of each of the components of a system (similar to the makefiles used by Make). Language-based dependencies (such as %include in C) are automatically detected and do not have to be explicitly stated as they must in Make. Using this database, DSEE supports two modes of development; a dynamic mode in which all changes made by one developer are immediately integrated into the environments of the other developers (Make operates in this fashion) and a cautious mode in which developers are protected from the changes of the other developers until the changes are explicitly requested.

The Configuration Manager remembers which versions should be used of each of the system components. This information is used to produce a configuration description showing which version of each component was used to produce the system. A build log is also maintained which shows what was built, who built it, when and why it was built.

The Configuration Manager also manages the various versions of the object files which are produced using a *derived object pool*. Object files in the pool are automatically deleted if they haven't been used in a long time. These facilities make it quite easy to switch development efforts between several versions of the system.

4.3. Task Manager

The Task Manager allows developers to describe changes to modules. These descriptions are linked not only to the modules that are changed (as is done with SCCS) but they can be linked together so changes which are related to the same enhancement (or bug fix) can be recorded as a group. This feature provides a great history for future reference. There is also a way to describe tasks which are related to a particular enhancement but have not yet been completed. These tasks will become part of an active "To Do" list which appears in a window on the screen. When these tasks are completed, they are recorded in a transcript file. These tasks are not restricted to programming tasks.

4.4. Advice Manager

The Advice Manager allows developers to document the procedure they used to perform a particular task (like adding a new option to a program). The task list is then available to others on the project who are interested in doing the same thing.

4.5. Monitor Manager

The Monitor Manager allows developers to place monitors (alarms) on particular modules so that whenever someone modifies that module a message is displayed. This message could be a warning concerning the dangers of modification or possibly a reminder to modify a related function in another module. The monitor also states who set it. If these monitors indicate that additional tasks need to be done, the tasks are added to the task list which appears on the user's screen.

4.6. Final Comments on DSEE

Future enhancements to the DSEE system will include syntax directed editors, interpretive debuggers and graphical program representations. The system seems well integrated and well designed. Unfortunately, it runs only on the Apollo Domain system running AUX.

5. What Should the Future Bring?

The CM system of the future should be highly-integrated into the development environment. It should at least have the integration and features of DSEE, the change request handling capabilities of CoCo, and the modification request support abilities of CMTS. It should also be available on a variety of operating systems (or at least many flavors of the same operating system in the case of UNIX). The system should not require graphics support.

Beyond all these capabilities, the ideal configuration management system should have a complete development method described in sufficient detail to allow a manager to develop a product using the tools provided. Although no development method will suit all situations, any model would be helpful. The lack of a well defined method for project use of SCCS and Make has hindered the use of these tools in many installations.

6. Final Comments

I have mixed emotions on the topic of configuration management. On the one hand I am amazed at the amount of published material on the topic (it's more than I expected); on the other hand, I am disappointed that a topic which is considered essential in hardware engineering and has been well defined in the software engineering for 25 years, is largely ignored in practice. As the practice of software engineering becomes more mature (scientific?), this is bound to change.

7. Acknowledgements

This paper was written for a class at the Wang Institute which was taught by Drs. Susan Gerhart and Alan Thompson. I would like to thank Gary Van Camp for the time he spent discussing configuration management as it applies to hardware engineering and also Manny Jasus, who explained the development method used at Bell Laboratories and gave me valuable feedback on the paper.

8. About the Author

The author is currently a member of a tools group at Wang Laboratories. Previously, he was the supervisor of the corporate tools group at Sykes Datatronics.

The author holds a Bachelors degree in System Software from the Rochester Institute of Technology and is pursuing a Masters degree in Software Engineering at the Wang Institute of Graduate Studies. He has published three other papers on software tools and techniques.

You can contact the author at Wang Laboratories, One Industrial Avenue, M/S 1989, Lowell Massachusetts, 01851, by phone at (617) 967-2609 or on UUCP at decvax!wanginst!wang!bazelmans.

References

[ADC 84] Software Maintenance and Development Systems, Inc.
Comparison, The ADC System and SCCS.
Advertising literature.
1984

[BAU ??] Bauer, H. A. and Burchall, R. H.
Managing Large-Stale Software Development with an Automated Change Control System.
IEEE , November, .

[BER 79] Bersoff, Edward H., Henderson, Vila D. and Siegel, Stan G.
Software Configuration Management: A Tutorial.
Computer :6-14, January, 1979.

[BER 84] Bersoff, Edward H.
Elements of Software Configuration Management.
IEEE Transactions on Software Engineering , June, 1984.

[BOE 79] Boehm, E.W., and Storm, N.E.
Change Management Tracking System Overview.
August, 1979.

[BRA ??] Bradford, E.G.
An Augmented Version of MAKE.

[CCC 84] Softool Corporation.
CCC vs SCCS.
Advertising literature.
April, 1984

[DEA 81] Dean, William A.
Why Worry About Configuration Management.
In Bryan, William, Chadbourne, Christopher, and Siegel, Stan (editors), *Tutorial: Software Configuration Management*. Computer Society PRESS, 1981.

[DEC 82] *CMS/MMS Code/Module Management System Manual*
Digital Equipment Corporation, 1982.

[DOL 76] Dolotta, T.A., and Mashey, J.R.
An Introduction to the Programmer's Workbench.
In *2nd International Conf on Software Engineering.* IEEE, October, 1976.

[ERI 84] Erickson, V.B., and Pelligrin J.F.
Build - A Software Construction Tool.
AT&T Bell Laboratories Technical Journal 63(6):1049-1059, August, 1984.

[FEL 78] Feldman, S.I.
Make - A Program for Maintaining Computer Programs.
AT&T UNIX related articles , August, 1978.

[GLA 78] Glasser, Alan L.
The Evolution of the Source Code Control System.
ACM Software Engineering Notes , November, 1978.

[GUY 80] Guyton, A.
Function and Use of an SCCS Interface Program.
April, 1980.
Update of Bonanni, March 1,1978.

[HUF 81] Huff, Karen E.
A Database Model for Effective Configuration Management in the Programming Environment.
In *International Conference on Software Engineering*, pages 54-61. IEEE, IEEE Computer Society, 1981.

[INC 84] Ince, D.C.
A Source Code Control System Based on Semantic Nets.
Software - Practice and Experience 14(12):1159-1168, December, 1984.

[JUA 79] Juanran, Joseph, J.
Quality Control Handbook.
McGraw Hill, 1979.

[KAS 82] Kasinskas, Joan W.
MESA - Management Environment for Software Administration.
April, 1982.

[KNU 76] Knudsen, D.B., Barofsky, A., and Satz, L.R.
A Modification Request Control System.
In *2nd International Conf on Software Engineering.* IEEE, October, 1976.

[LEB 84] Leblang, David B. and Chase, Robert P., Jr.
Computer-Aided Software Engineering in a Distributed Workstation Environment.
In *SIGPLAN/SIGSOFT Symposium on Practical Software Development Environments.* ACM, April, 1984.

[MCC 81] McCarthy, Rita.
Applying the Technique of Configuration Management.
In Bryan, William, Chadbourne, Christopher, and Siegel, Stan (editors), *Tutorial: Software Configuration Management.* Computer Society PRESS, 1981.

[MIL 68] *Configuration Control - Engineering Changes, Deviations, and Waivers*
Department of Defense, 1968.

[MIL 70] *Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs*
Department of Defense, 1970.

[ROC 75] Rochkind, Marc J.
The Source Code Control System.
IEEE Transactions on Software Engineering , December, 1975.

[STD 83] *IEEE Standard for Software Configuration Management Plans*
IEEE, 1983.

[TIC 82] Tichy, Walter F.
Design, Implementation, Evaluation of a Revision Control System.
In *6th International Conf on Software Engineering*, pages 58-67. IEEE, September, 1982.

[WET 82] Wetmore, Tom, and Paceley, Susan.
The Object Generation System.
March, 1982.