Software Configuration Management

SEI Curriculum Module SEI-CM-4-1.4

December 1990

James E. Tomayko
Software Engineering Institute

Carnegie Mellon University
Software Engineering Institute

This work was sponsored by the U.S. Department of Defense.

This technical report was prepared for the

SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

JOHN S. HERMAN, Capt, USAF
SEI Joint Program Office

This work was sponsored by the U.S. Department of Defense.

414122190

Software Configuration Management

Acknowledgements

I would like to thank The Wichita State University, particularly my department head Mary Edgington, and my dean Phillip D. Thomas, for making it possible for me to prepare the original version of this work at the Software Engineering Institute. Norm Gibbs, the Director the Education Program at the SEI, made sure I had the resources and encouragement to complete the work, including sponsoring a Configuration Management Workshop at the Institute.

Mark Chweh, a Carnegie Mellon student assistant, did the initial bibliographic search. Kate Harvey, another student assistant, kept track of the bibliography, provided editorial assistance, and wrote an excellent summary of the Configuration Management Workshop. Allison Brunvand and Oliver Martin helped prepare the slides and make the Workshop run smoothly.

My special thanks go to the participants in the Workshop. Brad Brown, Ted Keller, Dick Parten, and Bill Tindall, all from the corporate world, generously contributed their time and experiences. Jim Collofello and Bob Glass interrupted work on their own curriculum modules to participate. Mary Shaw, then the Chief Scientist of the SEI, also contributed.

Finally, my thanks to the reviewers of draft and later versions of this module, who generously contributed time and talent: Anita Baker, Ellen Borison, Lionel Deimel, Priscilla Fowler, Bob Glass, John Nestor, Joe Newcomer, Jim Perry, and Linda Pesante.

Contents

A support materials package, SEI-SM-4, is available for this module.

Software Configuration Management

Module Revision History

Version 1.4 (December 1990)	Minor corrections and format changes Approved for publication
Version 1.3 (July 1987)	Format changes for title page and front matter
Version 1.2 (April 1987)	Slight cosmetic changes
Version 1.1 (November 1986)	Added material to outline sections III.1, III.2, and to “Software Support” section
Version 1.0 (September 1986)	Original version

Software Configuration Management

Capsule Description

Software configuration management encompasses the disciplines and techniques of initiating, evaluating, and controlling change to software products during and after the development process. It emphasizes the importance of configuration control in managing software production.

Philosophy

Configuration management is an integral part of the software development process across all phases of the life cycle. It functions as a controlling discipline, enabling changes to be made to existing documentation and products in such a way as not to destroy the integrity of the software. Since configuration management extends over the life of the product, and since tools, techniques, and standards exist solely aimed at its proper execution, configuration management can stand alone as a module within a graduate curriculum.

The module presented here is intended to be an in-depth consideration of configuration management, including configuration item identification, change reporting and evaluation, change execution, tool evaluation and use, version control, and management principles related to configuration control.

Objectives

Cognitive Domain. The student will be able to:

1. Explain that software evolves throughout the life cycle by means of requirements changes (including new product definition) and discrepancy correction.
2. Define the term *configuration item.*
3. Explain how configuration control maintains the integrity of configuration items.
4. Explain that configuration items are components of the total product.
5. Identify the configuration items of a typical product.
6. Define the term *baseline.*
7. Explain the importance and use of configuration item nomenclature.
8. Differentiate between discrepancies and requested changes.
9. Differentiate between discrepancies caused by requirements errors and those caused by development errors.
10. List the key items included in a discrepancy reporting form.
11. List the key items included in a change request form.
12. Show how discrepancy reports and change requests are tracked within a software development organization until closure is obtained.
13. List and define the fundamental principles of implementing change control boards.
14. Define the scope of change control boards of at least three levels of a product development organization (system, subsystem, software).
15. Given the scope of a change control board, determine its composition.
16. List the most important considerations in evaluating the repair of discrepancies.
17. List the most important considerations in evaluating change requests.
18. List the most important considerations in evaluating requests for new derivatives of a product.

19. Specify how the implementation of changes can be tracked.
20. Define the simultaneous update problem.
21. Define the concept of *version trees*.
22. Identify at least three necessary characteristics of good version control tools (automated documentation, locking, minimal use of storage).
23. List at least three commercially available version control tools, their similarities and differences, and their suppliers.
24. Identify at least two standards for configuration management plans.
25. List the contents of an effective configuration management plan.
26. List at least three personal characteristics needed by effective configuration management personnel.
27. Describe the procedures of a configuration audit.
28. Explain the relationship between configuration management, quality assurance, and the customer.
29. Determine the impact of the granularity of configuration control on cost and schedule.
30. Explain the concept of *system description languages.*

Affective Domain. The student will:

1. Appreciate the role effective configuration management plays in ensuring product integrity.
2. Realize that configuration management activity extends over the entire software life cycle.
3. Be committed to implementing and demanding effective configuration management of software products.

Prerequisite Knowledge

The student should understand software life cycle models to the depth presented in a one semester undergraduate introductory software engineering course.

Prior experience in acting as a member of a team doing software development is also necessary.

Address of Author

Comments on this curriculum module are solicited, and may be sent to the SEI Software Engineering Curriculum Project or to the author:

James E. Tomayko
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Module Content

Outline

I. Introduction
  1. Evolution in the Software Life Cycle
  2. Configuration Management as a Controlling Tool
    a. Maintaining Integrity of Configuration Items
    b. Evaluating and Performing Change
  3. Configuration Management Process as a Visibility Tool: Status Accounting and Audits
  4. Configuration Management as a Cost Saving Tool
  5. Requirements for the Success of Configuration Management

II. Maintaining Product Integrity
  1. Identifying Configuration Items
  2. Establishing Baselines
  3. Naming Configuration Items

III. Change Management
  1. Types of Change
    a. Discrepancies
    b. Requested Changes
  2. Configuration Control Boards
    a. Fundamental Principles to Guide Configuration Control Boards
    b. Determining Configuration Control Board Characteristics
    c. Configuration Control Board Activities
    d. Executing Change

IV. Version Control
  1. The Simultaneous Update Problem
  2. Version Trees
  3. Tools for Version Control
  4. System Description Languages

V. Metrics

VI. Configuration Management Planning
  1. Content of Configuration Management Plans
    a. Scope
    b. Standards
  2. Characteristics of Personnel

Annotated Outline

Most material directly supporting this module can be found in [Bersoff80], [Babich86], and in "Summary of the 1986 SEI Configuration Management Workshop" (available in the support materials for this module, SEI-SM-4-1.0). Specific references are made to other sources where appropriate.

I. Introduction

1. Evolution in the Software Life Cycle

Software is evolutionary in nature. From the time a software product is defined until it is no longer used, it changes. Each change results in a different version of the product. Initiating, evaluating, and implementing the changes while maintaining product integrity is the purpose of configuration management. It provides a rational framework with which to deal with the irrational world of user demands and resource constraints. In terms of maintaining product integrity, it works closely with quality assurance and verification and validation teams.

2. Configuration Management as a Controlling Tool

a. Maintaining Integrity of Configuration Items

Changes to one configuration item usually require changes in others. For instance, a requirements change means specification, design, code, and testing changes. Configuration management helps maintain the integrity of specific items in an atmosphere of change.

b. Evaluating and Performing Change

Configuration control boards, established under a configuration management plan, evaluate proposed changes and discrepancy reports, authorize or do not authorize change, and track the implementation of the decision either way.

3. Configuration Management Process as a Visibility Tool: Status Accounting and Audits

By defining baselines and placing documents and other items under configuration control, configuration management provides project leaders and higher management with concrete evidence that a product is being created. Audits establish what the status of the project is at particular points in time.

4. Configuration Management as a Cost Saving Tool

By helping to maintain product integrity, configuration management reduces overall software development costs. Cost savings during a particular phase of the life cycle depend on the depth of application of configuration management. For instance, controlling individual source code modules costs more than only controlling the fully integrated product, but should result in overall savings due to reduction in side effects from individual changes. At this time, however, there are no quantitative measures sufficiently well developed to document the cost savings. This is largely because the losses due to lack of configuration management do not occur, and thus cannot be measured.

5. Requirements for the Success of Configuration Management

The key requirement for success of configuration management is the commitment of all levels of management to enforcing its use throughout the project lifetime. Configuration management, like other parts of software engineering perceived as being tedious, may require some coercion for success. A further requirement is the availability of a clearly stated configuration management plan.

II. Maintaining Product Integrity

1. Identifying Configuration Items

A *configuration item* is a document or artifact explicitly placed under configuration control. The minimum number of controlled items in a software project is whatever may be needed to effectively maintain and enhance the product. These may include requirements, specification, and design documents, source code, test plans, user and maintenance manuals, interface control documents, memory maps, and others such as procedural or policy documents. The actual items under control vary with the needs of the project, and certain items may be waived at specific points in the life cycle. Remember that there are time and cost tradeoffs associated with the number and level of items under control.

2. Establishing Baselines

In cooperation with the developing organization (or in small projects within the developing organization), configuration management helps to establish baselines by placing items under configuration control. Often the only discrete indication that a project has moved from one phase of the software development cycle to another is a controlled document signaling the end of the previous phase.

3. Naming Configuration Items

One aspect of configuration management is the naming of configuration items. Software products should be given unambiguous names and/or part numbers. Some guidance in naming conventions is given in military standards. Documentation prepared for a product is also numbered using the same scheme as the eventual product. (See [Gunther78].)

III. Change Management

1. Types of Change

a. Discrepancies

Discrepancy reports may be filed by anyone in the development organization, marketing, or by customers after delivery of the software product. Discrepancy reports are tracked by the configuration management organization through initial logging, scheduled evaluation by the configuration control board, and through the disposition process. Some companies have developed proprietary tools which prohibit linking modules with identified errors, thus preventing incomplete or erroneous software from becoming part of a shipped product.

There are three kinds of discrepancies that may be identified in reports.

(i) Requirements Errors

This type of discrepancy is an error in the requirements. Either the customer or marketing did not fully or clearly express the requirements, or incorrect information was given. (In the support materials package for this module, see the configuration management example's first discrepancy report for a requirements error.)

(ii) Development Errors

Another type of discrepancy is an error done during development. This means that a correct requirement was improperly implemented. Development errors occur between the time the requirements are baselined and the time the product is turned over to the customer or to marketing.

(iii) Violations of Standards

Yet another type of discrepancy is a violation of development standards, either the company standard or a customer standard in effect due to contract.

b. Requested Changes

Change requests are treated largely like discrepancy reports.

There are three kinds of changes that may be requested.

(i) Unimplementable Requirements

One reason for a change request is that a requirement turns out to be unimplementable through resource constraints identified by the requester. Another reason is that a "bad" implementation makes meeting all requirements impossible.

(ii) Enhancements

Enhancements are change requests that involve additional requirements.

(iii) Improvements

Improvements are change requests that will improve the product, though not in terms of functionality or performance. An example would be a request to rewrite a block of code to increase understandability.

2. Configuration Control Boards

a. Fundamental Principles to Guide Configuration Control Boards

(i) Principle of Authority

A configuration control board must have the authority to evaluate and direct the implementation of change to the part of the product within its scope of activity.

(ii) Principle of Solitary Responsibility

Many large development organizations have found that each board must have on it one person who makes decisions based on the advice of the other members. This is not a democratic process.

(iii) Principle of Specificity

The scope of responsibility of a control board must be limited to a predefined area of product development.

(iv) Principle of Responsiveness

Tardy consideration of items by a configuration control board undermines its operation. If the developers or customers feel that their reports are not taken into consideration promptly, they may try to circumvent the board or find another company to do business with. Simple acknowledgement of the receipt of reports is also important.

b. Determining Configuration Control Board Characteristics

(i) Hierarchies of Control Boards

Control boards are established for the total system, subsystems, and components of products. For example, an airplane with a digital flight control system would have a configuration control board responsible for controlling change to the entire aircraft, a subordinate board for the avionics subsystem, a further subordinate board for avionics hardware and software, and even further down the tree, boards for reused and new software. (See the viewgraph "Hierarchies of Configuration Control Boards" in the support materials package for this module.) The concept of hierarchies implements the Principle of Specificity.

It is also important to establish appeal routes for each level of the hierarchy, making the checks and balances explicit.

(ii) Scope

To implement the Principle of Authority, the scope of items that a particular configuration control board will consider must be specifically delineated. Boards are interdependent. A change to the software may result in a further change to the hardware, thus a higher-level board must approve. Changes to software that have no effects outside the software portion of the system may be considered by the software board alone. Documentation of the scope of the authority of the boards must be part of the Configuration Management Plan.

(iii) Composition

To implement the Principle of Solitary Responsibility, each board must have on it the person who is in a position to make and enforce the decisions of the board. The highest level board will always have the project manager on it. The software control board must have the principal software architect on it, assuming that person has the power to enforce changes. Additional members include those who can most effectively contribute advice to the sole decision-making person. The configuration manager responsible for the level of the project that the board controls should also be a member, and should have responsibility for tracking and implementing decisions.

c. Configuration Control Board Activities

The most important activity of a Configuration Control Board is the evaluation of discrepancy reports and requests for changes. Key factors considered by a board in making decisions are:

- Size of the change
- Complexity in reference to related systems
- Date when it is needed
- Impact on current and subsequent work
- Cost

- Criticality of the area involved
- Approved changes already in process
- Test requirements
- Resources (skills, hardware, system)
- Central processing unit and memory impact
- Politics (customer/marketing desires)
- Maturity of the change
- Is there an alternative?

Which of these factors to consider depends on whether the item being considered is a discrepancy or a request for a change or an enhancement.

(i) Evaluating Discrepancies

All reported discrepancies need not be fixed. Evaluating key factors may indicate that a particular error may not be economically repaired, so it is tracked as a waived item. (Also see viewgraph "Discrepancy Report Evaluation Process Flowchart" in the support materials package for this module.)

(ii) Evaluating Change Requests

Consideration of this type of change almost always requires additional time and cost evaluation. Enhancements need nearly the same evaluation criteria as those used in the initial decision to build the product.

d. Executing Change

Executing change involves delivering the authorized change document to the appropriate development organization, inspecting the change and its verification, modifying the affected configuration items, and recording the completion of the change by the configuration management organization.

IV. Version Control

1. The Simultaneous Update Problem

One of the most serious configuration management problems is that of simultaneous update, when two or more programmers are modifying the same portion of code. There is a distinct possibility that one person's changes will cancel or distort another person's, thus causing a software failure. Checking out code and other documents for modification must be handled by mutual exclusion, either manually (using a physical librarian) or automatically (using version control software).

2. Version Trees

Modifications to software create different versions, some of which must coexist. Reasons for this include incremental releases of limited products and forked development paths.

3. Tools for Version Control

Several automated tools for version control exist and are documented: SCCS, RCS, CMS, Cedar, Domain. These are described in [Rochkind75], [Tichy85], [DEC85], and [Leblang85]. Recently the Polytron Corporation released an adaptation of RCS for the MS-DOS microcomputer market.

4. System Description Languages

Current research [Borison86] is being conducted into System Description Languages. In some way, this is an extension of the *make* tool. Apollo's Domain system includes this concept. Basically, it is a way to accurately repeat the builds of a particular version of software.

V. Metrics

An important metric for configuration control board purposes is a trend analysis of discrepancy and change requests. Even simple counts of change requests and discrepancy reports will give an indication of whether side effects from fixes or changes are occurring. Also, some differentiation in terms of the severity of the errors should be exercised.

VI. Configuration Management Planning

1. Content of Configuration Management Plans

a. Scope

Configuration management plans need to be complete in all phases where configuration management is to be used in the project. Therefore, such plans are usually project-specific.

b. Standards

Many organizations have an existing standard for configuration management plans. The IEEE standard ANSI/IEEE Std 828-1983 [IEEE83] is an example of a generic standard.

2. Characteristics of Personnel

Configuration management personnel should ideally have software development experience, be conscientious, and be willing to inspire others to be conscientious!

Glossary

baseline

The point at which a document or other object becomes a configuration item.

configuration control

The process of managing change to a configuration item.

configuration item

A document or other object placed under configuration control.

discrepancy

An error in software caused either by improperly implementing a correct requirement or failing to implement it.

enhancement

A change to a product designed to improve or augment its performance.

Teaching Considerations

Textbooks

There are two book-length treatments of software configuration management. [Bersoff80] is the more complete, in that it provides a rationale for each concept of the subject. [Babich86] is slightly easier to understand and use, but is less complete.

Teaching Techniques

The following activities, when integrated with the syllabus for illustrative purposes, are effective in providing active experiences to complement the passive experiences in lectures:

- Review of case studies in class.
- Hands-on use of automated version control tools.
- Role-playing in simulated meetings of configuration control boards. Using an existing software product with which the students are familiar (such as the local operating system), have them write change requests and discrepancy reports and evaluate them according to the criteria presented in the module.

Teaching Support

The support materials package SEI-SM-4-1.0 contains a variety of materials to help in teaching a configuration management course.

It is also a good idea to contact the configuration management team (or person!) in a nearby software development organization, and visit with them about how they do their job. They will often loan copies of configuration management plans and other documents if you promise to black out the company name. Needless to say, these make great examples, even if they are terrible and have to be used as counterexamples. That configuration manager might also be coerced into leading one of your fake board meetings as the decision maker to add a touch of realism.

Software Support

Use of RCS, SCCS (both on UNIX), or CMS (on DEC VMS) is encouraged, as these are easily available. If no tools currently exist on your local system, use a software engineering project course to develop them. Kernighan and Pike, in their book *The UNIX Programming Environment* (Prentice-Hall, 1984), discuss the construction of a simple version control tool (pp. 165-170).

Suggested Course Adaptations

Government/Industrial Short Courses. Based on actual experience teaching the content of this module in the fall of 1986, a one-day short course on this subject can be constructed. The audience could be any combination of software engineers, project leaders, and technical managers. It is important to stress when presenting this material that the instructor is not acting as a consultant or a trainer, but rather surveying the current best practice of configuration management. The task for the class members, then, is to translate this knowledge into specific actions that can be taken to improve configuration management in their project/division/company. (Note that training and consulting activities can also be supported by this module, but the module was not written with either of those activities specifically in mind.)

Several days prior to the short course, it is helpful to distribute [Bersoff79], [Bersoff84], something on advanced configuration management environments and tools, such as [Leblang85], the elevator control example from the support materials package for this module, and copies of your viewgraphs. You might also want to construct a pre-test and post-test for evaluation purposes. Remember to inform the test subjects that their performance on the exams does not in any way affect their job ratings!

A schedule for the one-day workshop is as follows:

- First hour: Introductory material and Section I.
- Second hour: Sections II and III.1.
- Third Hour: Section III.2.
- Fourth Hour: Sections IV, V, and VI.

After the formal presentation of the material, the students should be divided into small (≤6 persons) groups in some logical fashion (by division, managers, etc.) and asked to consider the following questions. (If non-technical managers have sat in on the course, it is a good idea to ask them to leave if you feel that their presence would inhibit the discussion.) Instruct the groups that they will be expected to report their findings.

Where do you go from here?

Brainstorm the following questions:

- What are your goals re configuration management?
- What do you need to do to accomplish them?
    - tools
    - money
    - time
- What do you need to do first? Next? If you are a manager? If you are an engineer?
- How will you know when you have met your goals?

After about 30-45 minutes of brainstorming, the groups should reconvene and make their public reports. If you plan to be working with the organization to do follow-up, suggest that each group send you a one page summary of the results of their discussions. These can later be used to trace whether the groups implemented their own suggestions.

Before leaving the short course, each participant should be asked to fill out an evaluation of the course and of the instructor. Fundamental questions for the form include:

1. Did the course meet your expectations? Explain.
2. Please comment on the instructor.
3. Please comment on the handouts and readings.
4. What further information/help do you need?

One-Term Introduction to Software Engineering Course. Using part of the course materials in a one semester or quarter introduction to software engineering is highly recommended, especially if the course contains a project component. It is important that awareness of configuration management be established early in the course, certainly before any configuration items are produced. A one-hour presentation of the fundamental concepts (using examples of the differences between discrepancies and changes, and a sample configuration management plan) is usually sufficient for an understanding good enough to incorporate configuration management into the student project. The sample configuration management plan in the support materials package for this module was developed by students during such a course after the presentation outlined here.

A Software Engineering Seminar. The content outlined in this module is clearly insufficient in and of itself for a full quarter or semester course. As material regarding system description languages, additional tools, and large case studies is developed, this may change. For now, however, its use at the graduate level is probably limited to a few weeks presentation during a topics in software engineering seminar. The presentation in a seminar should be accompanied by case studies, or by a graduated example taught through the use of real software.

A Graduate Course on the Controlling Disciplines. The most effective use of this configuration management module is in concert with modules treating the accompanying controlling disciplines of software development: quality assurance and verification and validation. These topics are difficult to present in isolation from their partners. An integrated course consisting of the content of configuration management, quality assurance, and portions of testing that closely relate would fill a quarter or semester if actual experiences using tools and manual techniques on real software are included.

Bibliographies

There is only a very small body of literature on software configuration management. Configuration management of hardware systems such as aircraft has long been in effect and has a wider set of sources. In essence, much that has been written for hardware applies to software, and the reader is urged to look at sources outside the software field if time permits. (An excellent example is Samaras and Czerwinski [Samaras71]. Its Chapter 21 deals with software.)

This bibliography contains only those items judged to be of direct use to an instructor. A longer listing of version control and configuration management sources appeared in the July 1986 issue of *Software Engineering Notes* and is included in the support materials package.

Textbooks

Babich86

Babich, W. *Software Configuration Management*. Reading, Mass.: Addison-Wesley, 1986.

Though newer than [Bersoff80], this book is much more limited in scope. It does have sections using UNIX tools and ALS as examples.

Bersoff80

Bersoff, E. H., V. D. Henderson, and S. G. Siegel. *Software Configuration Management*. Englewood Cliffs, N.J.: Prentice-Hall, 1980.

This book contains the most complete description of software configuration management available. It provides a fairly complete rationale for what to do and why to do it. The authors have their own conceptual breakdown of the subject that does not map one-for-one with the organization of this module. The book also is weak in clearly explaining how to do the tasks of configuration management.

Gunther78

Gunther, R. C. *Management Technology for Software Product Engineering*. New York: John Wiley, 1978.

Pages 321-333 are useful, in that they contain a description of version numbering and naming schemes.

Samaras71

Samaras, T. T., and F. L. Czerwinski. *Fundamentals of Configuration Management*. New York: Wiley Interscience, 1971.

This book is now out of print. A bound, photocopied reprint may be obtained from University Microfilms International of Ann Arbor, Mich.

Other Materials

Bersoff79

Bersoff, E. H., V. D. Henderson, and S. G. Siegel. "Software Configuration Management: A Tutorial." *Computer 12*, 1 (Jan. 1979), 6-14.

This article contains the conceptual essence of the later book by the same authors.

Bersoff84

Bersoff, E. H. "Elements of Software Configuration Management." *IEEE Trans. Software Eng. 10*, 1 (Jan. 1984), 79-87.

Abstract: *Software configuration management (SCM) is one of the disciplines of the 1980's which grew in response to the many failures of the software industry throughout the 1970's. Over the last ten years, computers have been applied to the solutions of so many complex problems that our ability to manage these applications has all too frequently failed. This has resulted in the development of a series of "new" disciplines intended to help control the software process.*

This paper will focus on the discipline of SCM by first placing it in its proper context with respect to the rest of the software development process, as well as the goals of that process. It will examine the constituent components of SCM, dwelling at some length on one of those components, configuration control. It will conclude with a look at what the 1980's might have in store.

Borison86

Borison, Ellen. "A Model of Software Manufacture." In *Advanced Programming Environments: Proc. of an Intl. Workshop, Trondheim, Norway, June 16-18, 1986*, Reidar Conradi, Tor M. Didriksen, and Dag H. Wanvik, eds. Lecture Notes in Computer Science, vol. 244. Berlin: Springer-Verlag, 1986, 197-220.

Abstract: Software manufacture *is the process by which a software product is derived, through an often complex sequence of steps, from the primitive components of a system. This paper presents a model of software manufacture that addresses the amount of work that has to be done, after a given set of changes has been made, to consistently incorporate those changes in a given product.*

Based on a formal definition of a software configuration that characterizes a software product in terms of how it was manufactured, the model uses difference predicates *to discriminate between changes that are significant and those that are not. A difference predicate is an assertion about the relationship between two sets of components. Difference predicates determine when one set of components can be substituted for another. By predicting when existing components can be substituted for the output of a manufacturing step, difference predicates determine which steps in the manufacturing process can be omitted when incorporating a given set of changes.*

This is a formal presentation of the concepts of system description languages.

DEC85

Digital Equipment Corp. *VAX DEC-CMS Code Management System, Version 2.2*. Maynard, Mass.: Digital Equipment Corp., 1985.

Huff81

Huff, K. "A Database Model for Effective Configuration Management in the Programming Environment." *Proc. 5th Intl. Conf. Software Eng.* New York: IEEE, 1981, 54-61.

Abstract: *The effective management of configurations by programmers requires automatic techniques which are operative in the program development environment. In this paper, an abstract model is developed to cover the significant aspects of a typical programming environment pertinent to configuration management, using a database to capture configuration knowledge. The two aspects of the model deal with configuration identification and configuration control. In considering configuration identification, it is shown that the tools in the programming environment determine which configuration items need to be identified and also determine what the interesting and useful relations are among those items. In considering configuration control, the notion of a workspace, consisting of certain modification rights and certain visibility into the database, is developed to prevent conflict and to promote cooperation among programmers. The entire model can be used to evaluate the effectiveness of configuration management within a particular programming environment or as the basis of a programming environment design.*

IEEE83

IEEE. *IEEE Standard for Software Configuration Management Plans*. New York: IEEE, 1983. ANSI/IEEE Std 828-1983.

This is a cryptic standard listing recommended section headings without being a tutorial on content.

IEEE88

IEEE. *IEEE Guide to Software Configuration Management*. New York: IEEE, 1988. ANSI/IEEE Std 1042-1987.

Leblang85

Leblang, D. B., and G. D. McLean, Jr. "Configuration Management for Large-Scale Software Development Efforts." *Proc. GTE Conf. on Software Engineering Environments for Programming-in-the-Large.* Waltham, Mass.: GTE Laboratories, 1985, 122-127.

Abstract: *Teams working on very large software development efforts encounter equally large administrative problems maintaining and supporting their software, design, tests, and documentation. The DOMAIN® Software Engineering Environment (DSEE™) solves many of these problems. It is a distributed computer-aided software engineering environment that runs on Apollo® engineering workstations. A previous paper [by Leblang] described management of source code, documents, tasks, inter-project dependencies, and a basic configuration management system. This paper describes work that has occurred since that time. In particular, the full DSEE Configuration Management System (CM) is described. The DSEE CM is capable of building systems for multiple target machines while maintaining several concurrent configurations. It can distinguish between derived object modules based on the versions of the sources that were used to build them and the translator options used. The DSEE CM can track software releases in the field and relate released software back to the home office database.*

This paper, and others by Leblang in the supplemental bibliography, describe the Apollo DOMAIN software engineering environment and its built-in configuration management tools. Most of the advanced workstations in the immediate future will probably have configuration control tools similar to those reviewed here.

Rochkind75

Rochkind, M. J. "The Source Code Control System." *IEEE Trans. Software Eng. 1,* 4 (Dec. 1975), 364-370.

Abstract: *The Source Code Control System (SCCS)*

is a software tool designed to help programming projects control changes to source code. It provides facilities for storing, updating, and retrieving all versions of modules, for controlling updating privileges, for identifying load modules by version number, and for recording who made each software change, when and where it was made, and why. This paper discusses the SCCS approach to source code control, shows how it is used, and explains how it is implemented.

A description of the SCCS version control system packaged with UNIX.

Tichy85

Tichy, W. F. "RCS—A System for Version Control." *Software—Practice and Experience 15*, 7 (July 1985), 637-654.

Abstract: *An important problem in program development and maintenance is version control, i.e. the task of keeping a software system consisting of many versions and configurations well organized. The Revison Control System (RCS) is a software tool that assists with that task. RCS manages revisions of text documents, in particular source programs, documentation, and test data. It automates the storing, retrieval, logging and identification of revisions, and it provides selection mechanisms for composing configurations. This paper introduces basic version control concepts and discusses the practice of version control using RCS. For conserving space, RCS stores deltas, i.e. differences between successive revisions. Several delta storage methods are discussed. Usage statistics show that RCS's delta method is space and time efficient. The paper concludes with a detailed survey of version control tools.*

A description of RCS, an improvement over SCCS.