AUTOMATING CODE AND DOCUMENTATION MANAGEMENT (CDM)

Margaret E. Singleton
BTG, Inc.

AUTOMATING CODE AND DOCUMENTATION MANAGEMENT (CDM)

The Intelligent Guidance of Change

Margaret E. Singleton
BTG, Inc.

A Reston Book
Prentice-Hall, Inc.
Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

Singleton, Margaret E.
Automating code and documentation management (CDM).

A Reston book.
Bibliography: p.
Includes index.
1. Computer software. 2. Electronic data processing documentation. I. Title.
QA76.754.S56 1987 005.1 86-8112
ISBN 0-8359-9344-2

Cover design: Lungren Graphics Ltd.
Manufacturing buyer: Ed O'Dougherty

Cover art and Figure 5.1A have been adapted from an original drawing by Vicky Heim, and are used herein by permission of the artist

A Reston Book

A Division of Simon & Schuster
Englewood Cliffs, New Jersey 07632

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-8359-9344-2 025

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

FOREWORD

Many treatises have been written addressing documentation-, data-, code-, and configuration-management disciplines and their application within government and industry. Unfortunately, historical approaches and solutions to problems involving these disciplines have tended to fractionate instead of integrate them. Often, an organization's structure and policy arbitrarily inhibit the integration of documentation and configuration management disciplines that naturally should be integrated within that organization.

This text is a step toward the solving of this problem, providing ample evidence of the benefits that can be gained through integration, particularly when automation can facilitate that end.

There are really two disciplines addressed in this text, not four as alluded to above, for government and industry tend to use different terms for the same thing. Thus we find that the terms "data and configuration management" as used in government circles correlate with the terms "code and documentation management" as used in commercial data-processing circles.

Irrespective of the terminology used, the focus of this text is management of the products of the software development process. Clearly, these products are of different forms and representations, including specifications, manuals, source code, object code, data and media carrying the code, and data and documentation. Effective management of these products during their life cycle demands the application of the configuration management discipline to ensure a controlled evolution and systematic baselining of these products and, moreover, disciplined control of changes to them.

Automating the control of software products can greatly facilitate bringing together all of the products created by and flowing from the software development process. Awkward organization boundaries between documentation and software management disciplines can be overcome and even eliminated through automation, because of the integrating effect it has.

The reader will find that this straightforward, easy-to-read book draws considerably upon the experience and knowledge of others. With its topical style and organization, it should serve as a very useful guide and information

source for those entering the arena of automating, in whole or in part, the process of controling software products through configuration management.

Do not look to this book for theoretical expositions about the subject: This book is "down-to-earth," reflecting real experiences by real people and translating them into a software and documentation control methodology that can be easily grasped and applied using currently available automation techniques and tools.

VI HENDERSON
President, Desiderum, Inc.

PREFACE

This book has a single purpose: to demonstrate the cost-benefit of combining and automating software configuration management and software documentation.

When they see the term "configuration management" (CM), readers in the commercial sector may feel "that's for the military—our overhead would never support it." That's just the point! Automating CM and documentation, as proposed in this book, benefits you so much that you cannot afford *not* to do it. The problem of uncontrolled software and documentation now costs more than its solution.

Code and Documentation Management (CDM) is the integration and automation of two previously separate software disciplines: configuration management (CM) and documentation management (DM).

Software configuration management first appeared on projects developing large-scale software applications for the government. This discipline makes sure that no changes are made to the software without the cognizance or consent of everyone throughout the entire system or network.

Software documentation has usually been a separately organized process that takes place after the software is complete. This results in documentation that is usually out-of-date and therefore useless.

Code and Documentation Management (CDM) enables these two processes—software configuration management and software documentation—to occur simultaneously. The key to this accomplishment is automation, because computers are much better than people at all the drudgery involved in tracking changes. A single change to the software, like a stone dropped in a still pond, ripples to many different nooks and crannies within the software and its documents; it is difficult for any one person to know what the full impact of a change really is.

Chapters 1 and 2 of this book explain why you should use CDM. Chapter 3 is a graphic example of what CDM is. Chapters 4 through 6 explain how CDM affects the traditional software CM disciplines. Chapters 7 and 8 tell how to get started, for both centralized and decentralized applications.

In this book, "you" refers to anyone who wants to get better control of software and software documentation:

1. Managers and programmer/analysts involved with software development or maintenance projects.
2. MIS directors concerned with controlling data and text files in a distributed network of minis and PCs.
3. ADP directors maintaining and developing various applications for a large number of corporate users.

HOW TO USE THIS BOOK

If you are experienced in software configuration management and/or software documentation, read Chapter 1 for a fresh viewpoint, and Chapters 2 and 3 for new ideas. If at that point you like the concept, read the "Introduction" and "Recommendations" sections of the other chapters to know which ones you want to read entirely.

If you are new to these topics, congratulations! You have a chance to learn from, and avoid, the inefficiencies of the past. Please start with Chapter 1 and read everything.

ACKNOWLEDGMENTS

I would like to take this opportunity to thank all those who reviewed the manuscript: Gloria Gilliam, Gerry Lolmaugh, Les Garner, Barbara White, Ed Brooks, Harold Waddles, Julie Thomas, and Don Lewis.

The "display clutter" problem in Chapter 3 is based on an actual problem encountered and processed, using primarily manual methods, at SAIC.

I am indebted to Technology Training Corporation for the opportunity to teach a seminar on this subject; this book is based on my seminar material.

I wish to thank my colleagues at Ensco, Inc., Springfield, Virginia; Ford Aerospace and Communications Corporation, Hanover, Maryland, College Park, Maryland, and Houston, Texas; Science Applications International Corporation, McLean, Virginia; BTG, Inc., Vienna, Virginia; and Martin Marietta Corporation for their interest and support; and Martha Hook and the staff of Reston Publications for helping to get this book to publication.

Last but not least, I would like to acknowledge the tolerance of my children, Lucy and Lenny, who wondered if the book would ever be finished.

CONTENTS

ONE
WHY DO I NEED CDM?

1.1 INTRODUCTION

Code and Documentation Management (CDM) is the merger and automation of two related disciplines—software configuration management and software documentation. Taken independently, there are problems with trying to adhere to either of these disciplines. The problem with traditional, labor-intensive software configuration management (CM) is that as the size of the software increases CM becomes prohibitively expensive to do effectively. The manual process just cannot keep up with the overwhelming complexity and volume of the task at hand. The problem with traditional, labor-intensive software documentation is that the documents cost too much and are usually inaccurate, redundant, and out-of-date as soon as they are produced.

CDM, the merger of software configuration management and software documentation, is machine-intensive. It depends on automation: putting all software documents on-line and using such software tools as a text editor, a data dictionary/directory, and a database management system. Using this approach and appropriate tools, it is technically feasible to (1) quickly and accurately know how much a proposed change would cost to do correctly, (2) simultaneously implement changes in both the code and its documentation,

and (3) automate such necessary cross-references as the tracing of requirements.

CDM cuts costs, saves time, and provides better visibility and control than manual methods.

This chapter shows how the software development process works and how using the CDM approach—automated software configuration and documentation management—improves the quality of both software and documentation. It explains the software life-cycle, why CM and documentation are needed, and what the attendant problems are with doing things manually. It then provides an overview of the CDM approach.

1.2 THE SOFTWARE LIFE-CYCLE

Like human beings, software programs have a life-cycle. They are conceived (specification phase) and reared (design and code phases). After passing their college entrance examinations (unit tests), and then their college examinations (system tests), they are sent out into the world (operational phase). However, they still need—especially in today's constantly changing world—continued training (maintenance phase). These phases flow into one another, as is expressed by the "waterfall model" pictured in Figure 1-1. Different projects follow this model in different ways, of course. On large, formal projects, the documents are more voluminous because the software product goes through several reviews, one at the conclusion of each phase, before beginning the next. On small commercial projects the specification and design phases may be very informal.

Software development costs are discontinuous throughout this life-cycle. For instance, it costs more to maintain software than to develop it, as Figure 1-2 shows. Over a five- to ten-year life-cycle, maintenance costs *much more* than development. Although using the CDM approach produces benefits in all phases, it has its greatest impact in the maintenance phase.

A *baseline* is the software and its documents at a specific point in the development cycle. As the software develops it is constantly evolving, like a child growing. A baseline provides a "snapshot" of the software at a certain point in its development, like a student's school photograph. To "strike a baseline" means to place the software and its documents under configuration control, after which point changes can only be made with the approval of a change control board, and the entire baseline can be thoroughly reviewed.

There are three principal baselines: (1) functional, (2) allocated, and (3) product. All three baselines are shown in Figure 1-1. The functional baseline refers to the documents that identify which functions the software will perform. The allocated baseline assigns these functions to specific hardware and/

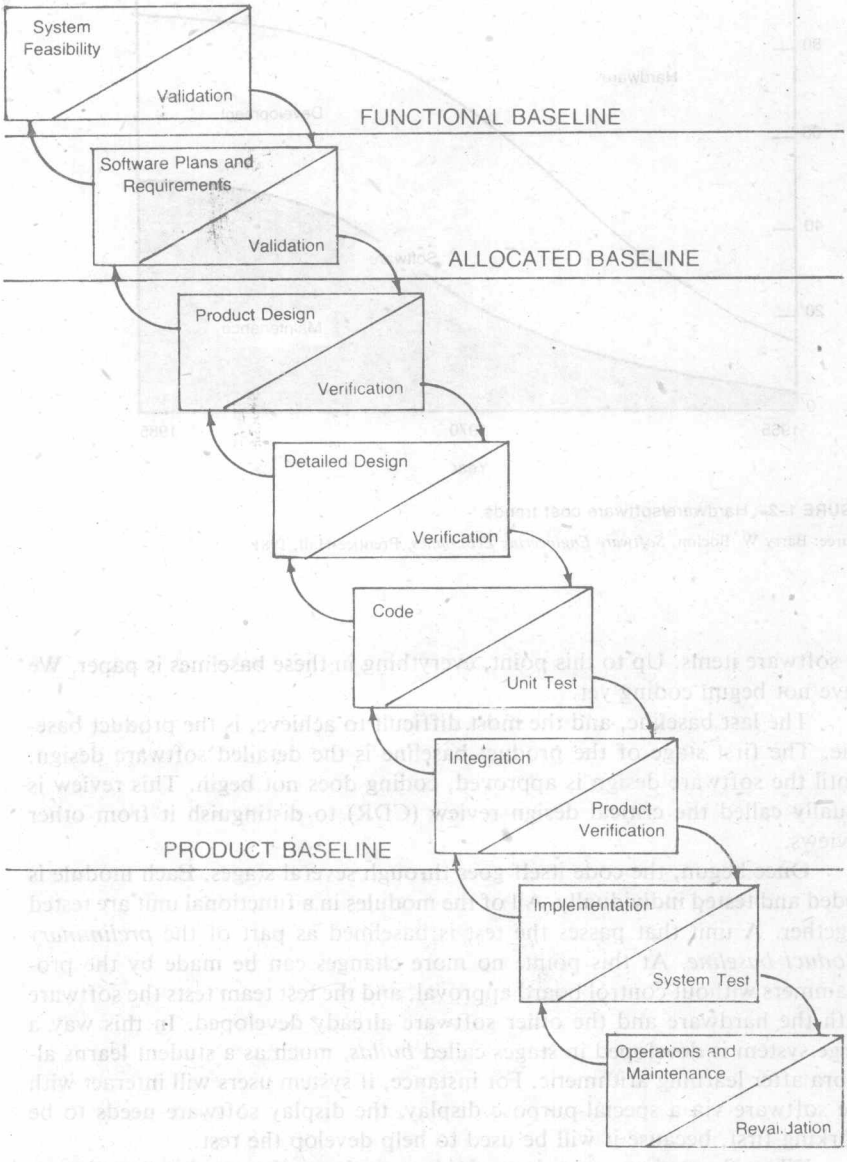

FIGURE 1-1 The waterfall model of the software life-cycle.

Source: Barry W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

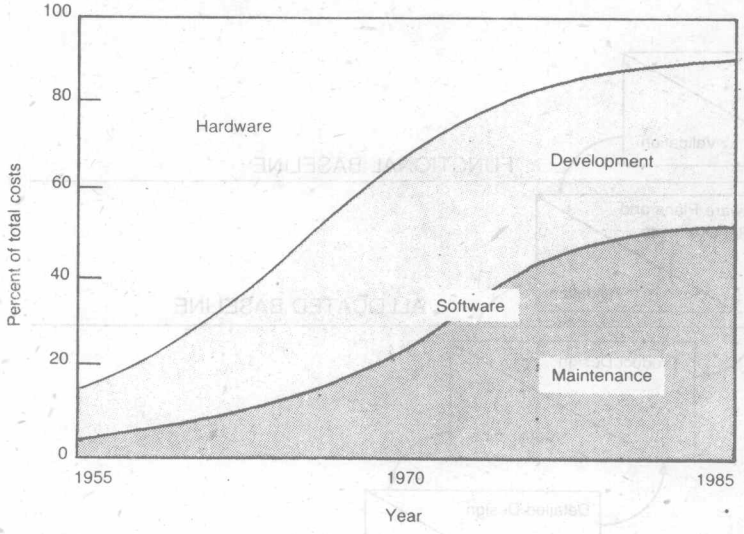

FIGURE 1-2 Hardware/software cost trends.
Source: Barry W. Boehm, *Software Engineering Economics,* Prentice-Hall, 1981.

or software items. Up to this point, everything in these baselines is paper. We have not begun coding yet.

The last baseline, and the most difficult to achieve, is the product baseline. The first stage of the product baseline is the detailed software design. Until the software design is approved, coding does not begin. This review is usually called the critical design review (CDR) to distinguish it from other reviews.

Once begun, the code itself goes through several stages. Each module is coded and tested individually. All of the modules in a functional unit are tested together. A unit that passes the test is baselined as part of the *preliminary product baseline.* At this point, no more changes can be made by the programmers without control board approval, and the test team tests the software with the hardware and the other software already developed. In this way a large system is developed in stages called *builds,* much as a student learns algebra after learning arithmetic. For instance, if system users will interact with the software via a special-purpose display, the display software needs to be working first, because it will be used to help develop the rest.

When the entire system is ready, it goes through operational testing to see if it can handle real-world problems. Resulting problems are fixed, and the *final product baseline* is delivered to the customer.

1.3 INTRODUCTION TO SOFTWARE CONFIGURATION MANAGEMENT

Software configuration management (CM) is a discipline that helps to control the development of software. It does this by structuring communication among software professionals about proposed changes to the software *before* these changes are made. Briefly stated, CM procedure is that anyone who wants to change baselined software first submits a change request to a change control board. Anyone whose work might be affected by the change is represented on this board. Only if the board approves is the change made. By developing such consensus and cognizance about proposed changes, CM procedures ensure that the new wrinkle that Marketing, for example, wants to put into their reports will not cause the Finance program to crash. In this way CM can help large-scale software development to take place, and make all software development a little easier.

CM procedures are needed because DP professionals tend to be independent by nature and are not likely to communicate so thoroughly without such procedures. In fact, research shows that the motivations of DP people are often different from the people in the rest of the company, as Figure 1-3 demonstrates. Here, "growth needs" refers to the need for new learning. "Social needs" refers to the need of people to be included with, approved of, and recognized by other people. People in management tend to be high on the social needs scale because they are strongly motivated to achieve status and results that other people will recognize. They wear the three-piece suit, and want visible signs of rank—a corner office, a company car, a six button telephone. On the other hand, the software development people tend to be high on the growth needs scale. Their desire is to continually upgrade their skills and to find or develop new tools to do the job better. The reason that they are attracted to software development is that they like to work by themselves and to puzzle things out. To them, the reward is in solving the puzzle: in getting the code to do useful things. Their expressed need for communication with, and approval from, other people is much less than that of people who work in different professions (sales, management, clerical, and so on).

The Story of Larry Walkman

To illustrate, consider a programmer whom I will call Larry Walkman, because he walked around the halls of an industry giant at all hours with a Sony Walkman plugged into his ears. At 4 A.M. on a snowy subzero Sunday morning in January, Larry's task was to complete his proposal volume and hand it to the word-processing operators. Larry, however, did not want to work with them because they were using magnetic card machines. This, he felt, was a terribly boring, antiquated way to do things. Instead, he keyed his document in to the mainframe in another building, formatting it there using

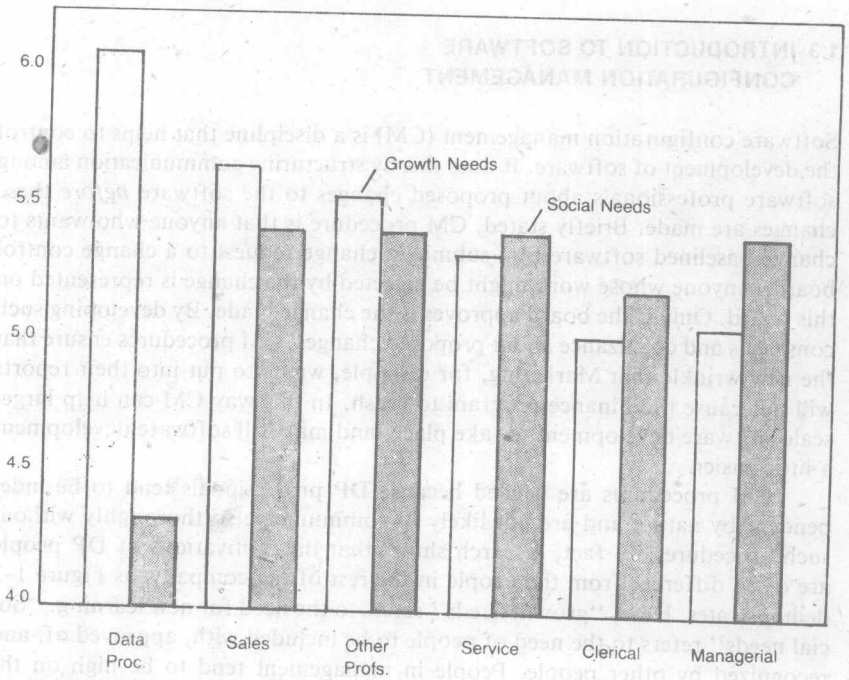

FIGURE 1-3 Comparative growth needs and social needs.

Source: Adapted from J. Daniel Cougar and Robert A. Zawacki, "What Motivates DP Professionals?" *Datamation*, Sept. 1978, pp. 117-128.

an IBM program called GML (Generalized Markup Language). Then he ran the formatted document back over to the laser printer in his building to print it out. Very jazzy.

When the laser printer started to print out, someone lifted the lid to see what was happening. In the resulting printout Larry then noticed an extra garbled line between lines 10 and 11 on one page. Functionally, it did not matter, because the print was legible and the typists were still going to enter this document on mag card anyway. All he had to do was walk the document down the hall, and he could go home.

However, Larry became fascinated with whatever was causing that extra line. He ran the text back through the mainframe, and printed it out on the laser printer again. This time, he did not touch the lid, but the extra line still appeared. Then he became *really* fascinated. He got on a terminal and called up the program that made the laser printer work. By now it was 5 A.M. The proposal volume assigned to Larry lay ignored on a table as he stared into the terminal, chasing the glitch he had discovered.