James Martin

A Carma McClure BOOK

MAINTENANCE
The Problem and Its Solutions

JAMES MARTIN
CARMA MCCLURE

PRENTICE-HALL, INC., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

Martin, James
Software maintenance.

Includes bibliographical references and index.
1. Software maintenance. I. McClure, Carma L.
II. Title.
QA76.9.S65M37 1983   001.64′2   82-25180
ISBN 0-13-822361-0

TO
CORINTHIA
AND
CONSTANCE

Editorial/production supervision by *Linda Mihatov*
Jacket design by *Diane Saxe*
Manufacturing buyer: *Gordon Osbourne*

Software Maintenance: The Problem and Its Solutions
James Martin and Carma McClure

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-822361-0

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
EDITORA PRENTICE-HALL DO BRASIL, LTDA., *Rio de Janeiro*
PRENTICE-HALL CANADA INC., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

SOFTWARE MAINTENANCE

A James Martin BOOK
Carma McClure

PREFACE

Software maintenance claims an extremely large share of the software dollar and is becoming the most expensive part of the software life cycle. Yet, although there are countless books and courses on systems analysis and design, the very important subject of software maintenance has been almost totally neglected. There is little understanding of what can be done to lessen the crippling maintenance problem.

In fact much can be done. Widespread use of the techniques described in this book would cut the maintenance costs in most organizations to a fraction of what they are today.

To solve the software crisis the tasks of developing, using, and maintaining software must be simplified and automated. Software technologies typically have focused on the development phases of the software life cycle—in short, programming methods and tools. Following the development of high-level programming languages in the early 1960s, the predominant software advance has been the introduction of software engineering, and in particular structured techniques. Unfortunately, we have grossly underestimated the need to change programs and the difficulties of doing so.

The cost of failing to design systems for maintenance is very high. Often this is calculated to be the overt cost of doing the maintenance. There is, however, a hidden cost which is often higher. The system becomes so fragile that programmers and their managers are reluctant to change it. Any change has unforeseen consequences which often cause problems in other parts of the system, annoy users, and consume precious software personnel time.

A fundamental problem with software maintenance is that when a change is made it often introduces unforeseen side effects. Fixing a bug has a great chance of introducing a new bug.

While the worst maintenance stories are nightmares, we should also

Over $20 billion per year are being spent worldwide on maintenance. If the techniques in this book had been employed everywhere they were appropriate, at least half of the $20 billion would have been saved.

CONTENTS

PART I INTRODUCTION

1 THE MAINTENANCE MESS

INTRODUCTION

We are accelerating into the computer age. Articles about computerization appear everywhere, from business journals to fashion magazines. By the end of the decade executives, secretaries, and consumers will all use computers as part of their daily activities, with an explosion of new applications such as electronic funds transfer, power facilities maintenance, exotic new weapons systems, automated design engineering, and robot production lines.

Yet when top business managers ask for information they know to be in their computers, they frequently cannot obtain it. When executives want to change procedures, they are told they cannot do that because the computer cannot make the change. Giant insurance companies have to resort to processing claims by hand after a change in government regulations. What is wrong?

The problem is that we have created computer programs that are very difficult to maintain.

WHAT IS MAINTENANCE?

We use the term *maintenance* to refer to changes that have to be made to computer programs after they have been delivered to the customer or user. We perform maintenance for a variety of reasons:

- To correct errors and design defects
- To improve the design
- To convert the programs so that different hardware, software, system features, telecommunications facilities, and so on, can be used
- To interface the programs to other programs

- To make changes in files or data bases
- To make enhancements or necessary changes to the applications

Program maintenance is different from hardware maintenance. Hardware maintenance for a computer consists of replacing deteriorated components, putting in engineering changes that correct defects and make design enhancements, and lubricating and cleaning mechanical parts. This does not affect how the computer is supposed to behave, so the user usually sees no change.

Program maintenance not only corrects defects and makes design enhancements; it also makes enhancements that change how the program behaves. Users constantly want to make adjustments in program behavior. Most maintenance work is caused by changing requirements rather than by reliability problems [1] (see Fig. 1.1).

Software systems used in industry are continually modified to adapt to changing data, and to meet changing user needs. Even a system that is totally reliable, completely meets user requirements, and is well structured will frequently be changed during the maintenance phase. Unless software systems of the future are designed to be changed more easily without jeopardizing their quality, maintenance of these systems will continue to be a time-consuming and costly activity.

If we push the analogy with hardware maintenance, the term "maintenance" seems inappropriate when referring to enhancements in program function. It might be better to use a different word. However, the word "maintenance" is now firmly embedded in common usage to mean all forms of software enhancement. Designing software for ease of maintenance

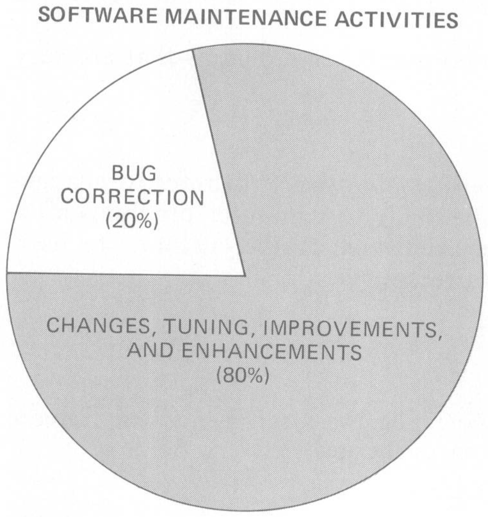

Figure 1.1 The major portion of maintenance work performed in most organizations is spent enhancing and improving software systems.

is the same problem whether we are talking about "engineering changes" or application enhancement.

INABILITY TO MEET APPLICATION NEEDS

Computers have plunged in cost dramatically. It is clear that the spectacular cost reductions will continue, accompanied by mass production of computers used in all walks of life. Can anything slow the momentum of the computer revolution? Certainly: ill-designed software.

Software has become the dominant factor in computerization. The success of computerization in many organizations hinges on the supply of software. The gap between the supply and demand of programs is rapidly growing. Most companies currently have a three- to four-year *backlog* of computer applications waiting to be programmed.

In addition, there is an *invisible backlog* of user needs which have not formally entered the queue of pending applications. A study by the Sloan School Center for Information Systems Research found that the invisible backlog of major enterprises studied averaged 164% of the declared backlog. The total backlog measured in this study represented 179% of the entire base of installed applications [2].

Programming accounts for an ever-increasing proportion of computer costs. Most software efforts of any magnitude are fraught with problems and failure. Programming projects take longer to complete and cost more than planned. As shown in Fig. 1.2, instances of actual costs running 300%

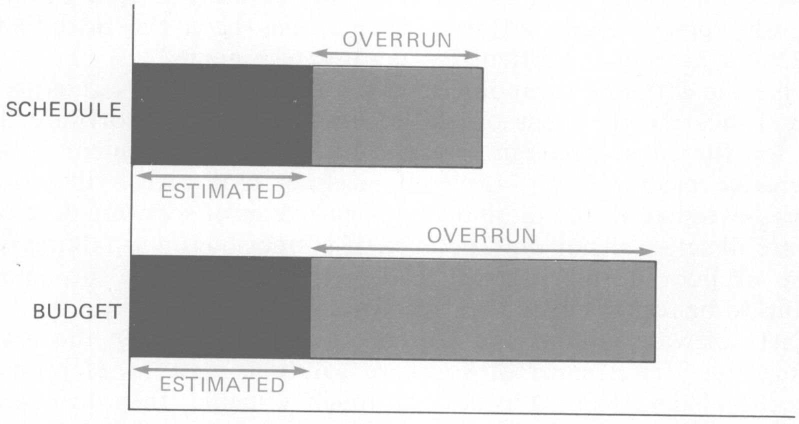

Figure 1.2 Instances of actual schedules running 200% over estimates and actual costs running 300% over budgeted costs are the rule rather than the exception in software projects.

over budget and actual schedules running 200% over estimates are the rule rather than the exception [3]. Managers are bewildered by their inability to apply normal management practices to the data-processing function. Users are frustrated and antagonized by applications that are difficult to change and do not work as expected. Software professionals are at a loss to understand why one project succeeds and the next one fails.

The economic stability of many nations is now being threatened by productivity lags. Computerization is seen as perhaps the best way to decrease production and office costs. Computer-aided design and manufacturing are rapidly being introduced. Major changes in paperwork procedures are needed. But the necessary changes need rapid increases in program production at the same time that there is an increasing shortage of analysts and programmers. A front-page *Wall Street Journal* article stated that "Oil and software are the two principal obstacles to economic progress" [4].

Advances in software technology have not kept pace with those in hardware technology. Installations that buy the latest computers often program in 20-year-old languages with ad hoc methods far removed from engineering discipline.

CLOSING THE SOFTWARE SUPPLY AND DEMAND GAP

Fundamental to the future of computerization is closing the supply and demand gap in programs. To solve the software crisis the tasks of developing, using, and maintaining programs must be simplified and automated.

Software technologies typically have focused on the *development* phases of the software life cycle—in short, programming methods and tools. Following the development of high-level programming languages in the early 1960s, the predominant software advance has been the introduction of software engineering, in particular structured techniques.

Like the software technologies of the 1960s and 1970s, those proposed for the 1980s mostly focus on the technical aspects of software development, for the most part ignoring the end user, management issues, and maintenance problems [5]. However, a closer look reveals that the latter problems overshadow the more technical problems of software development. Users are dissatisfied not only because of system bugs and failures but also because of poor documentation, inadequate training, and the inability of programs to be responsive to their changing requirements.

Past software failures and current cost trends explain the reason for doubting the effectiveness of software solutions that focus primarily on technical and development issues. Although valuable, they do not directly address a major cause of software crisis. Maintenance of existing software systems is diverting valuable and scarce resources away from new development efforts. It is a major contributor to the growing backlog of applications

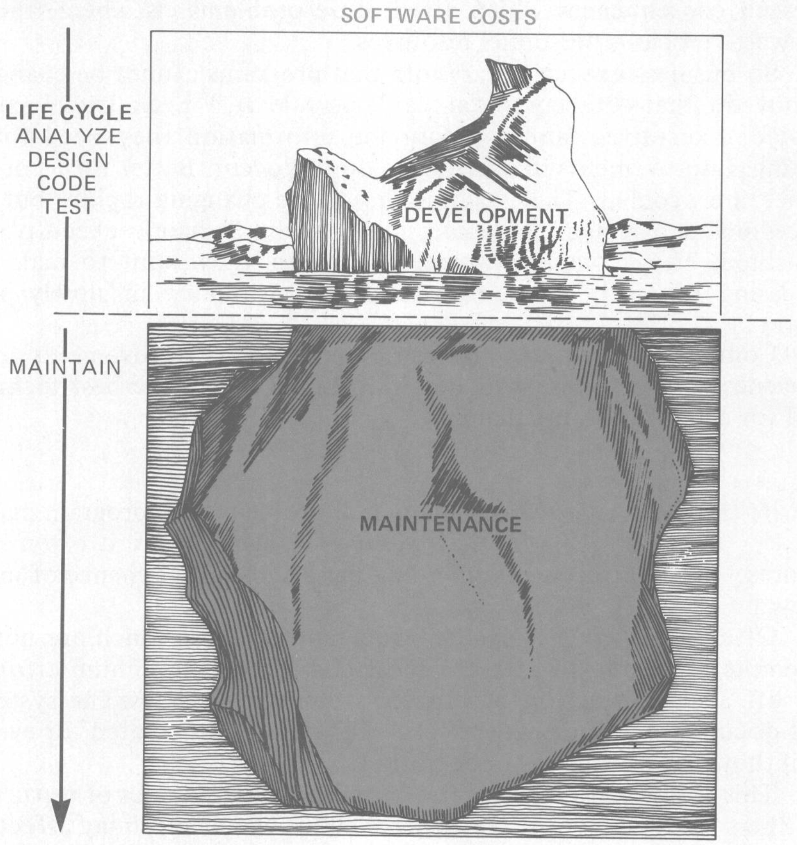

Figure 1.3 Software maintenance dominates the software life cycle. In many organizations, software maintenance activities consume three-fourths of the total life-cycle expenditures and over one-half of the data-processing personnel resources.

waiting to be programmed. Maintenance dominates the software life cycle in terms of effort and cost (Fig. 1.3).

We have grossly underestimated the need to change programs and the difficulties of doing so.

OPPORTUNITY COSTS

The cost of failing to design systems for maintenance is very high. Often this is calculated to be the overt cost of doing the maintenance. There is, however, a hidden cost which is often higher. The system becomes fragile so that data-processing managers are reluctant to change it. Any change has unforeseen consequences which often cause problems elsewhere, annoy users, and waste precious personnel resources.

So business executives are told that programs cannot be changed. "You cannot do that—the computer can't handle it." Even trivial changes are resisted. Executives cannot obtain the information they need for decision making. Improvements in procedures do not occur. Better forms of customer service are avoided. The business *should* be changing rapidly but the data-processing department is digging in its heels. Dynamic executives become frustrated. They constantly perceive changes they want to make but have increasing difficulty doing so. It is like swimming in slowly solidifying gelatin.

Computers offer the promise of enormous improvements in business efficiency. The promise will not be fulfilled unless the best techniques are used for achieving maintainability.

CHAIN REACTIONS

A fundamental problem with program maintenance is that when a change is made it often introduces unforeseen side effects. Fixing a bug has a substantial chance of introducing a new bug.

Often a change has system-wide ramifications which are not obvious. Attempts are made to affect a local change with minimum effort but this sets off a chain reaction of problems elsewhere. Unless the system is very well documented, these side effects will not be anticipated, or even known, until they cause problems in operation.

This situation is made worse because the maintainer or repairer is often not the person who wrote the original code. The change may affect the work of multiple coders.

Because of the side effects, maintenance needs far more program testing per line of code than other programming. When a change is made it may be necessary to run an entire bank of test cases to ensure that other areas still work correctly. Such *regression testing* to check for side effects can be costly. How costly depends on how convoluted the structure of the system is. In large entangled systems the chain-reaction effects can be very severe.

Maintenance changes tend to deteriorate the structure of programs, often making them more complex and more difficult to maintain next time. They are often done in an atmosphere of crisis. A quick patch is needed rather than any elegant restructuring. Patches accumulate. We get patches on top of patches on top of patches.

As flaw-fixing introduces new flaws, more and more time is spent on fixing these secondary problems rather than on correcting the structure that caused the original problem. The system steadily becomes less and less well ordered. Some complex systems reach a point where the maintainers cease to gain ground. Each fix introduces new problems. The system has become too unstable to be a base for progress.

Fred Brooks, philosophizing about his experience in managing the building of OS/360, wrote:

> Systems program building is an entropy-decreasing process, hence inherently metastable. Program maintenance is an entropy-increasing process, and even its most skillful execution only delays the subsidence of the system into unfixable obsolescence [6].

Lehman and Belady studied the history of successive releases of a large operating system [7]. It became steadily larger, the number of modules increasing linearly with the release number. However, the number of modules affected by maintenance changes grew *exponentially* with release number.

In corporate data processing (DP) the number of programs and files steadily grows. As this happens, the cost of maintenance tends to become a steadily larger part of the DP budget unless strong measures are taken to control it.

Figure 1.4 shows how maintenance has tended to grow in typical installations, becoming a larger portion of the DP budget. In many DP organizations, program maintenance activities consume nearly three-fourths of total life-cycle expenditures [8] and over half of the DP personnel resources [1]. Those organizations moving fastest into on-line and interactive systems are often spending as much as 80% of their time on maintenance. Using the types of techniques discussed in this book, this figure has been lowered to about 20% in some organizations.

A few organizations have reached a state where 100% of their effort is spent maintaining existing programs. No new applications are being

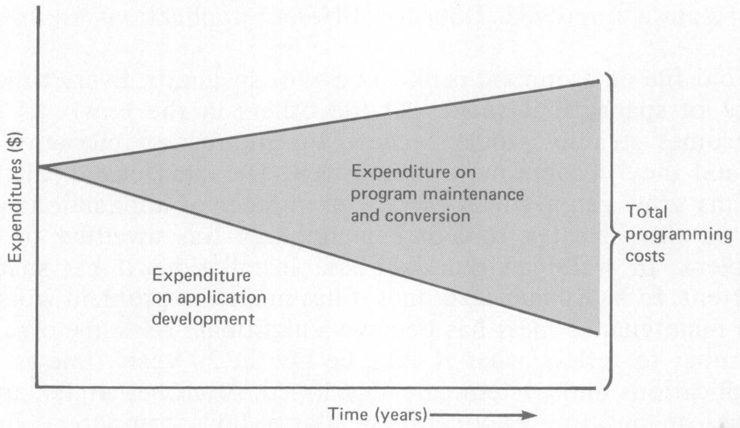

Figure 1.4 New application progress is often deferred by the rising cost of modifying existing programs and files. Some corporations spend more than 80% of their programming budget just keeping current and only 20% forging ahead.

written. One large government body froze all application development for 1½ years while it tried to redesign its data structures. Such a situation is intolerable for end users, who often find means to bypass the DP organization.

FILE PROBLEMS

Many DP organizations have built up vast libraries of files on tape or disk. When the business needs change, programs are often modified in a way that causes a change in the structure of a record. Unfortunately, some other program also uses the same record, so, somewhat by surprise, that program also has to be modified. In an old installation *many* other programs use the file and they all have to be changed.

A seemingly trivial change in a file environment sets off a chain reaction of other changes that have to be made. This upheaval is expensive and the necessary programmers are doing other work. Sometimes the modifications are difficult to make because the applications were not adequately documented.

As time goes on this problem becomes worse because more and more programs are created. More programs have to be changed whenever a file is changed.

It is often thought by systems analysts and data-processing managers that existing programs which work well can be left alone. In reality, however, the data that they create or use are needed for other applications and almost always needed in a slightly different form. New data-item types are added. New record types are needed with data-item types from several previous records. The data must be indexed in a different way. The physical layout of data is improved. Data for different applications are merged, and so forth.

An old file environment is like a bowl of spaghetti. Every time you pull one piece of spaghetti it shakes all the others in the bowl. As time goes by it becomes steadily worse because the number of pieces of spaghetti increase and they become more interwoven. The maintenance difficulties of file systems grow geometrically with the number of applications produced.

As we discuss later, data-base technology was invented to deal with this problem. In well-managed data-base installations it has succeeded to a large extent. In badly managed ones it has made the problem worse.

The maintenance mess has become a nightmare for some organizations. It is alarming to reflect what it may be like in 20 years time as more and more applications and systems are developed. Microbes under appropriate conditions can multiply exponentially like today's computers. But if they multiply when shut in an enclosed laboratory dish they eventually drown in their own excrement. One top DP executive compared this to his maintenance problems. New growth, he said, was being stifled by the ex-