SOFTWARE EVOLUTION

THE SOFTWARE MAINTENANCE CHALLENGE

Lowell Jay Arthur

TP31
A788-2

SOFTWARE EVOLUTION

THE SOFTWARE MAINTENANCE CHALLENGE

Lowell Jay Arthur

A Wiley-Interscience Publication
JOHN WILEY & SONS
New York / Chichester / Brisbane / Toronto / Singapore

Library of Congress Cataloging in Publication Data:

Arthur, Lowell Jay, 1951-
Software evolution.

"A Wiley-Interscience publication."
Bibliography: p.
1. Software maintenance   I. Title.

QA76.76.S64A78   1987   005.1′6   87-20972
ISBN 0-471-62871-9

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Software Evolution

Preface

In the software community the word "maintenance" has acquired a deadly negative connotation. An ugly word, "maintenance." It seems to imply that there is something desperately wrong with a software product (it must be someone's fault) and that the workers who slave over it are nothing more than blue-collar mechanics.

That's why this book is dedicated to software "evolution." As businesses grow, software must grow. As technology improves, software must evolve to match the technology. Every system—government, business, or personal—is in a constant state of flux, changing, growing. Software must *evolve* to meet the growing needs of these complex organizations and of people.

Software maintenance has been evolving at the same time as software. The five stages of maintenance maturity have been described as: chaos, concern, methodology, measurement, and control. This book will help you establish a repeatable maintenance process. You've probably already experienced chaos and concern. I also speak briefly about measurement. I developed these thoughts more fully in *Measuring Programmer Productivity and Software Quality,* published by Wiley, 1985. Somehow, I put the cart before the horse. Having the sort of quality control that delivers high-quality software productively is still a distant goal for most organizations. This book will help you reach the third level of maintenance maturity—a productive, high-quality, maintenance process.

Actually most software maintainers are involved in software evolution, not maintenance. In a typical software maintenance environment, corrective maintenance (fixing defects) consumes less than 10% of all resources. The rest focus on software evolution.

The software evolution process is shown in Figure 1. Evolution begins with a request for change and ends with the release of a software product.

As shown in the center of the figure, this book describes three types of maintenance: corrective (fixing software), adaptive (enhancing software), and one you may not have thought of, perfective (improving software quality). These are identical to the functions of an automobile mechanic: they fix problems, add radial tires or performance equipment, and change the oil or tune up the engine. Software maintainers often perform the first two but forget to take time out to fine-tune the software to prevent defects or to make it easier to enhance. Doing all three types *well* is essential to successful software evolution.

This book will describe what you need to do to establish a healthy, effective, evolutionary environment for supporting existing systems. Some of the key items are change control, system releases, configuration management, and software-maintenance management.

Chapters 7 and 8 will cover every aspect of perfective maintenance, from choosing software candidates to reengineering the software to improve its quality. These two chapters can teach maintainers and developers a lot about keeping software on the evolutionary ladder.

There is a method to this book's madness. It discusses the software-maintenance process in chronological order. Chapter 2 talks about entering a request for change, and Chapter 10 covers releasing the finished

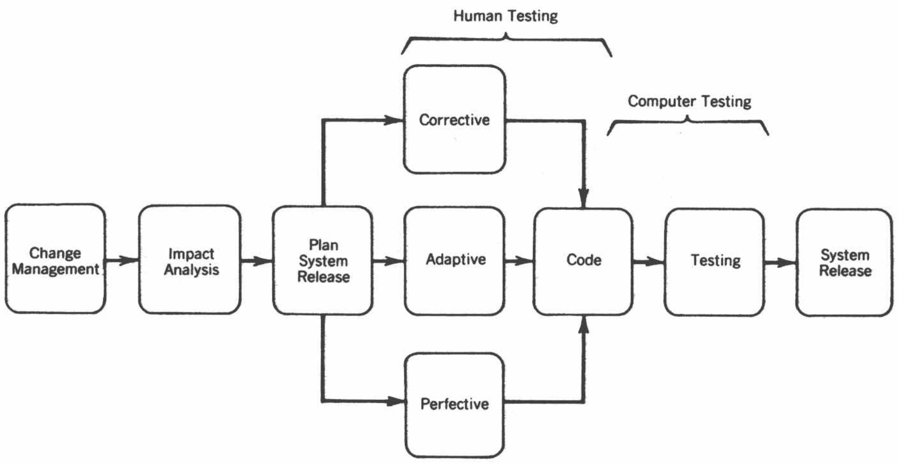

Figure 1 Software evolution process

product. Chapters 11 and 12 cover how to *implement* and *manage* a productive, high-quality maintenance environment.

I feel that a stationary maintenance environment is one headed for extinction. And that's what this book is all about: the evolution of software *and* the environment that supports it. They change constantly. Keeping up, or even getting ahead, is the software-maintenance challenge.

LOWELL JAY ARTHUR

Denver, Colorado
September 1987

To Tom Prieve,
Many thanks

Contents

CHAPTER

1

Software Evolution and Maintenance

In many ways software *maintenance* fails to describe the daily activities of the hordes of programmers and analysts who work on existing software. They constantly change software to meet the evolving needs of business, applications, and technology. In a typical environment these people actually spend less than 10% of their time fixing defects. They spend the majority of their time on enhancements—software evolution. Throughout this book you will see software maintenance and software evolution used interchangeably. *Software maintenance* means to preserve from failure or decline; *software evolution* means a continuous change from a lesser, simpler, or worse state to a higher or better state.

Because most organizations depend heavily on existing software systems, software maintenance is a critical function. Supporting these systems is the mission of the software maintainer.

To help you accomplish this mission, this chapter will:

- Explain the functions and flow of the software maintenance process.

- Define the three types of software maintenance: corrective, adaptive, and perfective.
- Identify the factors critical to successful, productive maintenance and evolution of software.

The major concern of software staffs today is how to maintain the existing portfolio of programs. Consider the following maintenance problems:

1. Most computer programs are difficult and expensive to maintain.

   One Air Force project cost $75 per line of code to build and $4000 per line to maintain.

   Software maintenance costs $300 billion each year worldwide, and demand is rapidly increasing (Martin 1983).

   Over the past 15 years the budget for maintenance has increased from approximately 50% of the resources expended on application software to 70–75%

   Each new development project adds to the maintenance burden. "Add little to little and you have a big pile!"

   End-user applications on micros, minis, and information centers will require maintenance.

   Demand for maintenance already exceeds the capabilities of most maintenance organizations. The user departments of businesses are programming many of their new applications. If maintenance is not managed and improved, demand will easily exceed available programming resources for both DP professionals and end users.

2. Software changes are poorly designed and implemented.

   Design documents are rarely examined and updated to reflect changes to the system.

   A carelessly planned system takes three times as long as estimated to complete; a carefully planned system takes only twice as long.

   Difficult-to-maintain systems are ultimately rewritten at great expense.

   The two years following the release of a new product are spent implementing enhancements to bring the system up to the user's *expectations*.

   Most major enhancements are so poorly understood and implemented that several additional releases are necessary to clean up the enhancement.

3. The repair and enhancement of software often injects new bugs that must later be repaired.

To resolve these problems and manage the growing software inventory, improvements are needed in the skills and productivity of maintainers, and in the quality and effectiveness of their work. This text focuses on helping you accomplish these goals.

To begin with, maintainers and managers should recognize that:

- Not all system maintainers are created equal, but they can be educated to equivalent skill levels.
- The difference between the best and worst performers is at least an order of magnitude.
- The reason for this disparity is a difference in the level of knowledge and skill, often referred to as *breakthrough* knowledge.
- The best performers can execute the key software maintenance activities more effectively than their counterparts.
- No single activity, or area of expertise, accounts for the differences.
- The key to maintenance productivity is to do most things a little better or faster (Peters 1985).
- A little more knowledge and skill multiplied over many activities produces striking differences in performance.

Providing maintainers with the lastest knowledge, skills, and techniques to achieve their mission by performing the key software maintenance activities a little better will reap significant productivity and quality improvements.

This text describes techniques for resolving many of the problems previously discussed. It describes the methods, tools, and techniques to improve your productivity and the quality of the software being maintained.

1. THE SOFTWARE LIFE CYCLE

The software life cycle covers the period from conception to retirement of a given software product. There are many definitions of the software life cycle. They differ primarily in the classifications of phases and activities. One traditional model is shown in Figure 1.1.

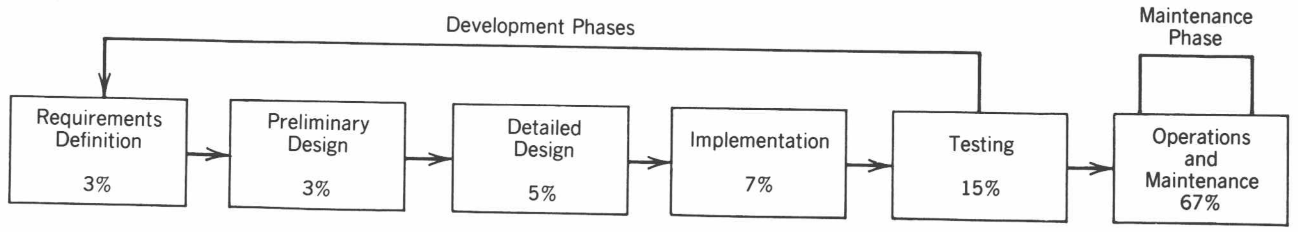

Figure 1.1 Software life cycle

As this diagram shows, for many large software systems, only one-fourth to one-third of all life-cycle costs are attributed to software development. The lions share of the effort and costs are spent in the operations and maintenance. (Note that the percentages indicate relative costs.)

2. SOFTWARE EVOLUTION ACTIVITIES

Software evolution consists of the activities required to keep a software system operational and responsive after it is accepted and placed into production. These activities include:

Correcting defects (maintenance)
Enhancing software functionality (evolution)
Improving the quality of existing software (maintenance)

In general, these activities keep the system in sync with an evolving, expanding user and operational environment. Functionally, software maintenance can be divided into these three categories:

Corrective *Corrective maintenance* focuses on fixing defects. Defects refer to the system not performing as originally intended, or as specified in the requirements. There are a variety of situations that can be described as corrective maintenance. Some of them include:

Correcting a program that aborts.

Correcting a program that produces incorrect results.

Corrective maintenance is a *reactive* process. Defects generally need to be corrected either immediately or in the near future.

Adaptive *Adaptive maintenance* includes all work related to changing how the software functions. Adaptive maintenance includes system changes, additions, insertions, deletions, modifications, extensions, and enhancements to meet the evolving needs of the user and the environment in which the system must operate. Adaptive maintenance is generally performed as a result of new or changing requirements. Some examples are:

Rearranging fields on an output report.

Changing a system to support new hardware configurations.

Adding a new function.

Deleting a function.

Converting a system from batch to on-line operation.

Making a program more efficient does *not* affect its functionality. As a result this type of change should be considered as part of perfective maintenance.

Perfective *Perfective maintenance* includes all efforts to improve the quality of the software. These activities can include restructuring codes, creating and updating documentation, improving reliability or efficiency, or any other qualities such as those discussed in Chapter 7 and 8. Some specific examples are:

Improving efficiency, maintainability, or reliability *without* changing functionality.

Restructuring code to make it more maintainable.

Tuning a system to reduce response time.

Although these three types of work are discussed separately in this text, much of the work is performed concurrently. For example, enhancements and quality improvements are often worked and tested together. Design of one program's changes will overlap the coding of another's. All of these activities occur during the software maintenance life cycle.

3. MAINTENANCE AND DEVELOPMENT DIFFERENCES

Although many activities related to maintaining and developing software are similar, software maintenance has unique characteristics of its own, including:

- *Constraints of an existing system.* Software maintenance is performed on an existing production system. Any changes must conform or be compatible with an existing architecture, design, and code constraints. Typically, the older the system, the more challenging and time consuming the maintenance effort becomes. Later chapters will discuss methods of preventing software extinction.
- *Shorter time frames.* Software development may span one or more years, whereas maintenance may span a few hours to cycles of one to