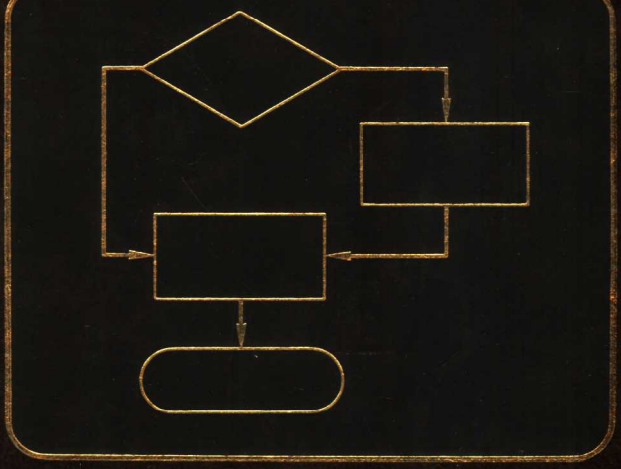
CONVERSION OF COMPUTER SOFTWARE

John R. Wolberg



CONVERSION OF COMPUTER SOFTWARE

John R. Wolberg
Technion-Israel Institute of Technology

Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

Wolberg, John R.
Conversion of computer software.

Includes index.
1. Software compatibility. I. Title.
QA76.6.W63 001.64'25 82-5253
ISBN 0-13-172148-8 AACR2

Editorial/production supervision and interior design
by *Anne Simpson*
Manufacturing buyer: *Gordon Osbourne*

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-172148-8

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Editora Prentice-Hall do Brazil, 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*

To the men of Plugah Zayin.

וַיֹּאמֶר יְהוָה הֵן עַם אֶחָד וְשָׂפָה אַחַת לְכֻלָּם וְזֶה הַחִלָּם
לַעֲשׂוֹת וְעַתָּה לֹא־יִבָּצֵר מֵהֶם כֹּל אֲשֶׁר יָזְמוּ לַעֲשׂוֹת׃

(בראשית יא 6).

Preface

The conversion of computer software is a task which most computer industry professionals approach with a mixture of fear and distaste. The decision to convert is usually the alternative of last resort, taken only when all other alternatives have been rejected. Managers hesitate to authorize conversions for fear of massive cost overruns and/or failure to meet schedules and performance objectives. Analysts and programmers usually approach conversion work with a clear lack of enthusiasm, as the work is often boring and repetitive and sometimes maddening.

Nevertheless, the need to convert software grows from year to year. The U.S. General Accounting Office issued a report in September 1977 stating that the federal government was spending more than $450 million to convert programs. Estimates of the worldwide annual cost of conversion is several billion dollars. Some surveys have estimated that the cost of conversion may run as high as 10% of the computing budget.

The purpose of this book is to give the reader an insight into the fundamental concepts of software conversion. The subject is considered from several different points of view:

1. The manager faced with a decision to convert or not to convert
2. The project manager faced with the task of organizing and managing a conversion project
3. The analyst and programmer involved in the details of conversion
4. The analyst and programmer assigned the task of developing conversion aids and tools

Not all subjects are of equal interest to all readers. For this reason the book has been organized so that the various chapters can be read independently. Each chapter includes an introduction and a summary to help the reader decide if the material in the chapter is relevant to his or her needs.

The first three chapters consider the subject from a broad point of view. The conversion process is considered in general terms in Chapter 1. Chapter 2 is devoted to the economics of conversion. Although the cost per line of a conversion varies considerably from case to case, there are a number of general rules that are applicable for all conversions. The management of a conversion project is discussed in Chapter 3. It should be emphasized that many of the well-known concepts regarding the management of software development projects are not completely applicable to conversion projects.

Chapter 4 considers the subject of program enhancements. Portability, performance, and maintainability are discussed in this chapter. The importance of this subject is often overlooked. When the primary objective of a conversion project is to move software from one system to another, a secondary objective can be to improve the software.

The next two chapters are devoted to technical solutions and strategies for conversion. Chapter 5 considers conversion software. Chapter 6 discusses conversion algorithms. The subject of conversion algorithms is really the technical cornerstone of a conversion project. Decisions related to the mapping of programs from one dialect or language to another have a profound influence on the performance of the resulting software.

The final chapter (Chapter 7) presents elements required in a language used for automatic converter development. The CONVERT language is used to illustrate the concepts developed in this chapter.

Throughout the book an attempt is made to use examples from real conversion projects. Examples have been taken from a variety of languages, including COBOL, FORTRAN, PL/1, BASIC, RPG, and Assembler languages. Other examples are related to data conversion and conversion of job control language programs.

Acknowledgements

I have been involved with software conversions since 1975. During this period my experiences with a number of people helped me formulate the opinions and concepts expressed throughout the book. Although I take full responsibility for the contents, I would like to acknowledge the roles played by some of these individuals.

My introduction to conversions resulted from a project initiated by Marshall Rafal of O.L.I. Systems and Kate Kalin of NCSS. We are still working together on a number of interesting projects. I spent several years involved in a variety of conversion projects with Ehud Huberman and Avi Peled, formerly of Shahat Ltd. They both contributed to my understanding of the conversion business.

My work with Alex Hill and Alan Reiter of Reiter Software Systems gave me insight into a number of topics related to the world of software. Through my contacts with Jeff Fenton, formerly of BSO Minisystems, and Bram de Hond of ARA Automation, I have had the opportunity to learn about the conversion marketplace in Europe. I would like to thank Jaap van der Korst of Philips for making it possible for me to spend several weeks at Plan Q seeing how a really big conversion project is managed.

I would like to acknowledge the useful material and insights I received from Nahum Rand of Rand Information Systems and Bob Dinkel of Dataware. Reference is made to this material in the book.

I have worked with a number of people developing conversion software and have learned from their experiences. In particular I would

like to acknowledge Eric Hulsbosch, Barbara Lasky, Bernie Roizen, and Adam Rosenzweig.

Some of the material in this book is the result of my own research in the area of software conversions. I would like to thank the Samual Neeman Institute for Advanced Research in Science and Technology for their support of this effort.

The typing of the manuscript was performed by Miriam Beatson, Marion Gold, and Sheila Herskovits. I would like to acknowledge their help and offer my thanks.

I would also like to thank the Technion for all the help I have received throughout the period that I have been involved in this work. A number of my fellow Technion staff members have read sections of the book and have offered useful suggestions.

Finally, thanks to my family for their patience and understanding throughout the duration of this project.

John Wolberg
Haifa, Israel

Contents

PREFACE *xi*

ACKNOWLEDGEMENTS *xiii*

CHAPTER 1 THE CONVERSION PROCESS *1*

1.1 Introduction *1*
1.2 Software Portability *6*
1.3 The Development of Programming Languages *11*
1.4 Program Conversions *19*
1.5 Complete System Conversions *29*

CHAPTER 2 CONVERSION ECONOMICS *39*

2.1 Introduction *39*
2.2 Conversion Estimation Equations *40*
2.3 Reprogramming and Redesign Estimation Equations *42*
2.4 Comparing Conversion, Redesign, and Reprogramming *45*
2.5 Optimizing a Conversion Project *47*
2.6 A Mathematical Model for Conversion Optimization *52*
2.7 Conversion Cost per Line *56*
2.8 Summary *58*

CHAPTER 3 MANAGEMENT OF A CONVERSION PROJECT *60*

3.1 Introduction *60*
3.2 An Overview of Conversion Management *61*
3.3 Preconversion Tasks *66*
3.4 Management Alternatives *68*
3.5 Conversion Planning *71*
3.6 Estimation Techniques *75*
3.7 Managing a Very Large Conversion: Plan Q *78*
3.8 Summary *86*

CHAPTER 4 PROGRAM ENHANCEMENTS: PORTABILITY, PERFORMANCE, AND MAINTAINABILITY *88*

4.1 Introduction *88*
4.2 Enhancing Portability *89*
4.3 Enhancing Performance *105*
4.4 Enhancing Maintainability *133*
4.5 Summary *146*

CHAPTER 5 CONVERSION SOFTWARE *149*

5.1 Introduction *149*
5.2 Program Standardizers *151*
5.3 Documentation Tools *157*
5.4 Automatic Converters *161*
5.5 Data Generators *169*
5.6 Data Converters *171*
5.7 Test Validation Software *172*
5.8 Project Management Tools *173*
5.9 Summary *174*

CHAPTER 6 CONVERSION ALGORITHMS *176*

6.1 Introduction *176*
6.2 Defining Patterns *178*
6.3 Some Simple Examples *185*
6.4 Blank Removal *188*
6.5 Identifier Names *193*
6.6 Missing Features *197*
6.7 Comparison of Character Type Items *201*
6.8 Data Files and Access Methods *205*
6.9 Summary *208*

CHAPTER 7 A CONVERSION LANGUAGE *210*

7.1 Introduction *210*
7.2 Language Architecture *212*
7.3 Pattern Recognition and Searching *215*
7.4 Controlling Logical Flow *216*
7.5 EXCLUDE: A Pattern Recognition Feature *220*
7.6 Changing a Line *221*
7.7 Usage of Tables *223*
7.8 Symbolic Parameters *225*
7.9 Other Features *227*
7.10 A Complete Conversion Rule *228*
7.11 Summary *231*

INDEX *233*

1

The Conversion Process

1.1 INTRODUCTION

1.1.1 Terminology

The need or desire to move software from one environment to another is fundamental to computer usage. The move might be triggered by a new hardware configuration, a new operating system, or language and compiler changes. Three basic alternatives are usually considered whenever a move is contemplated:

1. *Emulation* is a process whereby the new environment is made to directly execute software that was written for the original environment.
2. *Conversion* is a process in which changes are made in the software so that the original system will execute properly in the new environment.
3. *Replacement* of software is the most radical choice. Alternative software is either developed or obtained for the new environment.

If emulation is possible, no software changes are required. If the conversion option is selected, the original software is used as the starting point from which the new software is developed. Replacement of software implies that the original software is discarded entirely. Replacement can be accomplished by the purchase or leasing of standard

software or by a new development effort. We thus see that *conversion* is the middle ground between *emulation* and *replacement.*

When a decision has been made to convert the original software or develop new software, there are four basic strategies for completing the task:

1. *Translation* refers to the primarily automatic conversion of software.
2. *Recoding* refers to the manual conversion of software.
3. *Reprogramming* implies a software development effort which may include some system redesign but no significant functional redesign.
4. *Redesign* implies a software development effort which includes a functional redesign of the system.

Translation and recoding utilize the original software as the primary specification for the new system. Reprogramming and redesign yield software that bears very little resemblance to the original system. Reprogramming is usually a cheaper process than redesign because the old system does not have to be redesigned from a functional point of view.

Regarding terminology, in this book the *conversion* of software implies an important degree of translation and/or recoding. Some modules in the system might be reprogrammed and might even be redesigned; however, the bulk of the effort will be based on the original source code. Thus a distinction is made between systems that are converted, reprogrammed, or redesigned. In Chapter 2 we will see that this distinction has important economic implications.

1.1.2 Changes in the Computer Environment

The incentive for changing a particular computer environment is usually a combination of the following:

1. Reduced cost
2. Improved performance
3. Increased reliability
4. Increased capacity

In some situations the need for change can be caused by a problem such as discontinuation of support for a specific piece of hardware or soft-

ware. Whatever the reason for change, there is a price associated with this line of action. A decision to change the existing environment (by either hardware or software modifications) must include an analysis of the impact the change will have on the existing application software systems.

For some situations the impact on the applications software will be nonexistent. For example, aquisition of additional disk drives rarely affects the existing software. Alternatively, some changes will have a major impact on the software. For example, replacement of one computer by another computer from a different manufacturer might require a major conversion, reprogramming, or redesign effort.

In analyses of the desirability of change, it is standard practice to list the benefits and costs associated with the new environment. The decision is then based on answers to several questions:

1. Will the new environment answer present and projected needs?
2. Are the estimated benefits associated with the change commensurate with the estimated costs in both hardware and software?
3. If a conversion, reprogramming, or redesign effort is required:
   a. Are suitable personnel available on an in-house basis?
   b. Can the project (or projects) be subcontracted?
   c. What are the estimated durations for the proposed projects?
   d. What computer resources will be required for completing the projects?
   e. What are the cost estimates for the proposed project (or projects)?

There is a degree of risk associated with undertaking major configuration changes. In particular, if the applications software is to be altered, estimates of effort, computer resource requirements, and project duration are subject to error. Gross *underestimates of required resources* to complete the software changes can result in large cost overruns and delays. Clearly, as the size of the software system increases, the risk associated with cost overruns and delays increases.

Another risk associated with changes in configuration is connected with the *predicted performance* of the applications software on the new system. If the old software is compatible with the new environment, a benchmark can be run to measure performance on the new system [1]. However, if a conversion, reprogramming, or redesign effort is required, the predicted performance cannot be totally verified as a phase of the feasibility study. Nevertheless, if the major demands on computer resources can be identified in the original software, some intelligent

benchmarking can be used to reduce the probability of unpleasant surprises.

1.1.3 Methodology for Software Modification

Software can be modified by conversion or reprogramming. Some aspects of the methodology for affecting these types of changes are similar. If the software is to be completely redesigned, the methodology is similar to the well-known techniques for developing new software. Clearly, for both alternatives the methodologies are dependent on the system size.

Reasonable procedures for modifying small systems are not reasonable for large systems. For larger systems the degree of planning and coordination must be increased. The task of gathering, cataloging, and controlling the large volume of required material (e.g., programs, files, listings, flowcharts, manuals, etc.) becomes increasingly difficult. Testing procedures for large systems with many alternative paths through the system are clearly more complicated than the procedures required for smaller (and usually simpler) systems.

The actual modification process often complicates normal operations of a computer center. If the demand on computer resources is significant (when compared to total available resources), *scheduling problems* are encountered. For modification of large systems, the need for intensive testing can often significantly increase the normal work load. For critical software systems (i.e., systems that must remain in operation throughout the entire transition period), some degree of *parallel operation* must be anticipated as a part of the acceptance procedure. As a result of delays in software modification, the original budgeted time for parallel operation might be inadequate. If one is forced into the need to run two parallel computer installations for an extended period, large cost overruns are inevitable.

Many of the problems associated with software modification are the result of a lack of experience. Modifying existing software can be different from developing new software. Before undertaking a major software modification effort, it is extremely important to plan the effort intelligently and to be sure that the available personnel are capable of completing the modification successfully. A feasible alternative is to contract the work to a company specializing in the *conversion*, *migration*, or *transformation* of software. (The major companies providing software modification services often use different terms to describe their services. Conversion, migration, and transformation are probably the three most popular terms in use.)

1.1.4 Conversion versus Reprogramming

It has been noted that both conversion and reprogramming are methods for modifying software. There are similarities associated with these methods of *migration* (i.e., moving applications from one environment to another):

1. Both methods start with the same functional description of the application.
2. From the user's point of view, the software appears to be essentially the same before and after the migration. Whether the migration has been accomplished by conversion or reprogramming is of no interest to the end user.
3. For both methods, operating procedures associated with the application remain essentially the same before and after the migration.
4. For both methods, reports generated as part of the application remain essentially the same before and after the migration.
5. Since the software is only a black box (from the user's point of view) with defined system inputs and outputs, the testing procedures for conversion and reprogramming can be essentially the same.

Nevertheless, a basic difference exists between these two methods of software modification. Conversion technology utilizes the original software as the primary system documentation and starting point. Reprogramming is based only on the original functional description of the application, and the original software is essentially discarded.

The methodology for affecting a particular conversion can be described to a great extent algorithmically (e.g., when we see *X*, then do *Y*). Thus conversions can to a large degree be automated. Automation has several important effects on the conversion process:

1. The effort per line tends to decrease with an increasing number of lines. Since cost is usually proportional to effort, the cost per line also tends to decrease with an increasing number of lines.
2. Automation allows most of the conversion effort to be performed by lower-level personnel than are required for a reprogramming effort.
3. The conversion process is usually faster than comparable reprogramming efforts.

These differences lead to important cost advantages for conversions as compared to reprogramming as the size of the original system increases (see Section 2.4). The tendency is thus to convert rather than reprogram large systems when this option is possible. For some situations the conversion alternative is simply not feasible. For example, sensitive real-time applications programmed in an Assembler language and utilizing specific hardware features of the original system are usually reprogrammed if the move is to an entirely new environment. However, for most applications written in higher-level languages, the conversion option should be seriously considered before initiating a reprogramming effort.

1.2 SOFTWARE PORTABILITY

If software was truly portable, the need for conversions would be considerably reduced. The term *portable software* refers to software that can be moved from one computer environment to another with a minimum effort. Fenton defines portable software as software that can be moved after transformation using automatic converters [2]. The computer industry has long recognized the value of portable software; however, how one goes about achieving this goal is a debatable issue. An in-depth review of this subject is included in a book edited by Brown, *Software Portability* [3].

1.2.1 The Portability of Computer Programs

The most important step toward achieving portability of programs is to develop a standard language definition. The first attempt at language standardization was made by the American Standards Association [4]. Their X3.4.3 Committee was formed in 1962 to develop an American Standard FORTRAN and succeeded in issuing two standards in 1966 [5,6]. The first COBOL language standard was completed in 1968 [7].

COBOL is by far the most popular business-oriented programming language, and FORTRAN is the most popular scientific- and engineering-oriented language. Since both languages have been "standardized" for many years, it is useful to consider the impact of their standardization on language portability.

The standardization processes resulted in compromises between what the users wanted in the languages and what the manufacturers