Every Engineer Should Know/14

WHAT EVERY ENGINEER SHOULD KNOW ABOUT

MICROCOMPUTER PROGRAM DESIGN

Keith R. Wehmeyer

What Every Engineer Should Know About

Microcomputer Program Design

Keith R. Wehmeyer
Cincinnati Milacron
Cincinnati, Ohio

MARCEL DEKKER, INC. New York and Basel

Library of Congress Cataloging in Publication Data

Wehmeyer, Keith, [date]
What every engineer should know about
microcomputer program design.

(What every engineer should know ; v. 14)
Includes index.
1. Microcomputers--Programming. 2. Structured
programming. I. Title. II. Series.
QA76.6.W435 1984 001.64'2 84-12052
ISBN 0-8247-7275-X

MARCEL DEKKER, INC.
270 Madison Avenue, New York, New York 10016

Current Printing (last digit):
10 9 8 7 6 5 4 3 2 1

PRINTED IN THE UNITED STATES OF AMERICA

PREFACE

The computer has become the single most powerful problem - solving tool in today's technically minded world. Whether it is a pocket calculator or a large mainframe, the computer offers the ability to perform complicated, tedious tasks with great speed and efficiency. The birth and continued development of the microprocessor has made computing power available at reasonable cost to the home and small business.

A new problem therefore results: that of programming the computer to get the desired operations to perform properly. This book presents the concept of Structured Program Design, a systematic way to create computer programs efficiently. This method begins with an assessment of the problem and continues through the development, coding, and testing of the computer program. Built into this system is a method of programming that makes additions, enhancements, or corrections much easier to implement as the program undergoes revisions. The book is intended for two audiences: beginning programmers and experienced programmers seeking ways to improve the quality of their software. Structured Program Design techniques provide an excellent way for novice programmers to "think through" a problem until they arrive at a working solution. Advanced programmers in general do not presently use methods to improve coding efficiency or readability, areas where these techniques are of great help.

This book covers the entire scope of computer programming and Structured Program Design, from problem identification to maintaining existing programs. Chapter 1 presents a general overview of the Structured Program Design process, with subsequent chapters detailing each phase. An example is carried out through the book to show how each phase is implemented. An unusual feature of this book is that all the techniques presented here will work on a variety of computers that use many different languages. Thus, these techniques work as well on programmable calculators as they do on large business computers. The objective is still the same: to write programs that efficiently produce reliable output and are easy to use and understand.

Other supplemental areas of programming are covered such as a software library, programming personnel, and program documentation. These areas are often overlooked but play a key part in organizations with an ongoing programming effort.

I wish to express my thanks to Dr. William H. Middendorf, Professor of Electrical Engineering, University of Cincinnati, for his advice and editorial suggestions. My wife Jannis has been invaluable, both with her encouragement and typing skills. I am also indebted to my brother Stephen for his excellent artwork. And finally, a thank you to my entire family, whose support made this work possible.

Enjoy the book! I hope it makes your programs easier to write and maintain by reducing the headaches and late nights along the way.

Keith R. Wehmeyer

ABOUT THE AUTHOR

Keith R. Wehmeyer is currently employed as a Research Engineer in the Software Development group of the Robot Research department at Cincinnati Milacron Industries Inc., Cincinnati, Ohio. He is responsible for the hardware and software development of new robot control architectures. He received the B.S. degree in Electrical Engineering from the University of Cincinnati in 1982, and is currently pursuing a Masters of Science degree in Computer Engineering.

CONTENTS

1

AN INTRODUCTION TO MICROCOMPUTER PROGRAM DESIGN

This chapter presents a complete overview of microcomputer program design, with key topics presented in more detail in subsequent chapters. In addition, a structured technique used in writing programs is presented as it pertains to structured program design. This technique will be discussed throughout the following chapters, as it is the basis for good program design. Finally, the titles and responsibilities of several important people in any programming organization are discussed. We begin now with a discussion of what microcomputer program design is and why it should be used.

STRUCTURED PROGRAM DESIGN

Structured program design (SPD) is a systematic procedure used to create, test, and verify computer software. As in any other design procedure, the programmer works through a sequence of pre-planned steps until the project is completed. This technique offers many advantages to those who use it fully, some of which are as follows:

Efficiency. The programmer who proceeds under any set of defined guidelines should produce code quicker and with fewer errors. This reduces costs and can prevent many of the difficulties that will be discussed later in this book. As an example, Daly (1) conducted a study that compared software (any computer program) and hardware (the physical parts of a computer) projects of nearly equal complexity. His results showed that the software project took twice as long and cost four times as much to design and maintain as the hardware project. He attributed a large part of this increased time and cost to the fact that the hardware engineers used a more systematic approach toward design.

Maintenance. It is widely accepted that about 75 - 80% of the programmer´s time is spent maintaining existing corporate software.

Software maintenance differs from hardware maintenance in that software is "repaired" in order to correct errors or add functions, instead of fixing a system that no longer functions correctly. By using good design techniques, the programmer can free himself to spend more time on new design challenges. In addition, a program written using structured program techniques will be much easier to understand and modify.

Cost Reduction. Hardware costs drop a factor of ten every decade, so an increasing amount of a project´s total cost will depend on software generation. While the programmer may not be able to boost productivity at the same rate, any improvement will have an increasingly significant effect in controlling cost.

Unfortunately, many programmers do not use guidelines, and few are learning to use them. In a study by McClure (2), the results pointed to the fact that most programmers had not changed their approach to programming in the last five years and had no plans to do so in the next five. Thus, a great deal of improvement is possible by following only the simplest of guidelines.

PARALLELS WITH ENGINEERING DESIGN

Virtually every engineer is familiar with the engineering method of design. This basic approach stems from the scientific method, which is comprised of the following steps:

1. Observe a phenomenon.
2. Postulate a theory to explain the occurence of the phenomenon.
3. Construct a test to prove the theory.
4. Draw conclusions as to the validity of the theory based on the test results.

The engineering method parallels this with the following steps:

1. Recognize and specify a need.
2. Specify a product to fill the need.
3. Design the product according to the above specification.
4. Verify that the product design meets the specifications and fills the need.

Note that all four steps in the engineering method could be used to construct a program. Ideally, a specification should first be written based on a set of requirements. Next, a design process should be used to create and debug a program. Finally, the program should be checked against the specification for accuracy and completeness.

Other steps should be included in the software design process as well. Since most programs undergo frequent revisions and modifications, a convenient way of recording and documenting changes should be included. In addition, user documentation should be prepared, such as instruction or operation manuals. This documentation is very important, since it may be the only link between the program´s authors and users.

THE EIGHT STEPS OF STRUCTURED PROGRAM DESIGN

As previously stated, many of the steps used in other forms of engineering design are found in program design. Myers (3) summarizes the work of software generation into the following eight steps:

1. Requirements Analysis and Definition. This is the point where the user and the authors begin the design process by deciding what they wish to do with a given configuration of hardware. Notice that the function of the program is to control the hardware in an agreed-upon fashion.

2. Specification. At this point the desires of all parties are put into written form and are concretely defined. Time and cost limits are to be established and detailed as well. Since the specification will be referred to throughout the rest of the design process, the creation of a clear, well - defined specification is essential. This is usually the last point where a user´s input is considered until the program is operational.

3. Design. Once the program specification is done, the programmer then begins to determine what resources will be required. Following this, construction of a project workbook begins. This workbook should contain the specification and all other materials used in the project. Next, the program logic is contructed using one of several design techniques into a flow chart of operation. This flow chart can be one that uses actual code, English phrases, or symbols to denote what the program will do and when.

4. Programming. Once the program´s logic has been charted, it is up to the programmer to convert the flow chart into actual program code. Included in this conversion should be documentation showing the "how and why" of program operation. In essence, the flow chart statements are placed next to the code that performs the corresponding functions. Following this, a structured "walk-through" review of the program is suggested as an error-trapping mechanism before the program is entered into a machine. The scope and function of the structured walk-through is discussed later in this book. Finally, the testing and debugging process continues inside the machine until the program functions as its author believes it should. It is in this phase that most proponents of structured