DATA STRUCTURED PROGRAM DESIGN

Kirk Hansen

Ken Orr & Associates, Inc.

DATA STRUCTURED PROGRAM DESIGN

Kirk Hansen

Prentice-Hall, Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

HANSEN, KIRK (date)
Data structured program design.

Bibliography: p.
Includes index.
1. Electronic digital computers—Programming.
2. Structured programming. 3. Data structures.
(Computer science) I. Title.
QA76.6.H3338 1986 005.1'13 85-25729
ISBN 0-13-196841-6

Editorial/production supervision and interior design: Tracey Orbine and Diana Drew
Cover design: 20/20 Services, Inc.
Manufacturing buyer: Gordon Osbourne
Editorial consulting: Karen Howard Brown

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-196841-6 025

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*
Whitehall Books Limited, *Wellington, New Zealand*

Foreword

By the end of the 1970s, the business of designing software was dominated by four competing schools of thought:

- the *Constantine-Myers-Yourdon-Stevens School:* people who believe a system should take its shape from the pattern of interactions among functions performed by the system
- the *Warnier-Orr-Jackson School:* people who believe a system should take its shape from the structure of the data that drives it
- the *Parnas School:* people who believe a system should be structured so that its modular walls serve to isolate and conceal complexities of data and process
- the *Mugwump School:* people who believe design is for sissies and that the structure of the system should be whatever occurs to the coder while seated at the terminal.

As a result of my writing, I have long been classified as a member of the first of these groups, the "function-structured" school of design. And Kirk Hansen is clearly allied with the second group, the "data-structured" school of design. So, how have I come to be writing a foreword to Kirk's book?

My answer stems from a growing conviction that the first three approaches are all valid, and *all necessary*. The notion of using one of these disciplines to suppress the others is onerous; it amounts to saying, "Truth is so valuable we have to use it sparingly." For a given class of programs, any of the techniques might serve well. But if you're building a great diversity of applications, you need a diversity of approaches. If you're building *systems*, particularly large and complex systems, you need all the help you can get—a trick from Myers here, some Jackson "backtracking" there, a Warnier/Orr approach wherever the data gives a strong hint about process structure, some Parnas thinking to abstract and conceal complexity . . . and more.

That famous software engineer Mao Tse Tung said, "Let a thousand blossoms grow, let a thousand schools of thought contend." Of course, Mao didn't really mean it: He not only suppressed competing schools of thought, he even suppressed flowers, judging them to be "bourgeois." But I do mean it. I believe that as a software builder you need an army of ideas to support you. In particular, no matter what school you belong to, you need the ideas set out in Kirk Hansen's crystalline presentation of *Data Structured Program Design*.

Tom DeMarco
Principal
The Atlantic Systems Guild

A Note from Ken Orr

New ideas, even great new ideas, often take a very long time to become popular. One reason is because it is not enough that an idea be a good one, it must also be communicated to those who need it. Time after time, poor ideas that are well communicated have won out over good ideas poorly presented.

Kirk Hansen has given us the best of both worlds in this book. He has taken a great idea (data structured programming) and communicated it superbly. Not only has Kirk done an excellent job of communicating both the letter and spirit of the Ken Orr Data Structured Systems Development (DSSD®) methodology, he has been able to compare and contrast it with Warnier's and Jackson's approaches to data structured programming as well.

This book is a major contribution to the growing body of software engineering literature. It is clear and informative and, best of all, it is fun to read. Kirk Hansen has demonstrated conclusively that important technical ideas don't have to be boring.

One final note. Throughout the book, Kirk refers to the Ken Orr methodology. In this regard, it is important to point out that the Ken Orr methodology is not the work of a single person, or even that of a single firm. Rather, it is the result of the work of hundreds of people over the last two decades, including Mr. Hansen. I am personally delighted that someone as talented as Kirk Hansen should add his own particular brand of creativity to communicate the DSSD® methodology to a wide audience eager to make programming simple, straightforward, and ultimately scientific.

Ken Orr
Topeka, Kansas

Preface

You can derive the structure of a program from the structure of its data.

The insights that let you do this come from three people: Jean-Dominique Warnier, Michael Jackson, and Ken Orr. This book fully describes Orr's program design techniques and outlines the ideas of Warnier and Jackson.

This book is mainly about how to design programs, not how to code them, so it won't make much difference what programming language you know. But you should be familiar with at least one of the major procedural languages: PL/1, COBOL, FORTRAN, ALGOL, BASIC, Pascal. Where code is needed we use COBOL, since it's the most widely used language among business programmers and is fairly clear even to programmers who have never seen it before. The programs in this book have been compiled with the IBM OS/VS COBOL compiler and run on an IBM 3081.

The book will be more valuable if you keep a paper and pencil handy and try things for yourself as they are discussed. "I hear and I forget. I see and I remember. I do and I understand." It's easy to think you understand something when you're just watching it happen; the only way to know for sure is to try it yourself.

Several chapters contain suggestions for "further reading." These references give author and title only; full publication data may be found in the bibliography at the end of the book.

Finally, an explanation of the title may be in order. Data structured program design is a **structured** technique (a detailed procedure) for deriving the **design** of a **structured program** from the **data structure**. Shuffling the emphasized words gives "data structure structured structured program design," which is accurate but unpronounceable; so one of the structures has swallowed the rest.

NOTES ON THE REVISED EDITION

This Revised Edition has three main changes. First, three sections have been added. They are 6.4, "Hidden Hierarchies"; 14.6, "Using Hidden Hierarchies for Efficiency"; and 14.7, "The Logical Data Structure."

The second change is revision of some terms to match the latest release of Orr's DSSD® method. Specifically:

Logical Process Structure (LPS)	is now	Logical Output Mapping (LOM).
Logical Read Routine (LRR)	is now	Physical Input Mapping (PIM).
Physical Put Routine (PPR)	is now	Physical Output Mapping (POM).

The phrase "read routine" has mostly been changed to "input mapping," but the terms are often used interchangeably.

The third change is the separation of the material on inversion from the main flow of the book. Two principles guide the discussion of inversion in this edition:

1. Don't invert unless you have to. If you're lucky enough to work with a system that makes inversion unnecessary, skip the material on inversion completely.
2. Even if your system makes inversion unavoidable, ignore the inversion material on your first reading of the book.

To this end, Chapter 11 and Part III have been prefaced with suitable warnings, and the chapters in Part III have been reorganized so the inversion material is easier to skip.

ACKNOWLEDGMENTS

This book was the idea of Morris Nelson, vice-president of field operations at Ken Orr and Associates, Inc. I am grateful for his strong support and assistance in bringing it to fruition.

The book borrows heavily from KOA instructional materials, especially for examples and exercises. It owes a great deal to all the KOA instructors, past and present, who have contributed to the instructional material and to the development of the methodology itself. I am especially grateful to Ken Orr, Rick Messinger, and Stiles Roberts for their comments on an earlier draft. Thanks also to Dave Higgins, whose new book, *Designing Structured Programs*, impelled me to clarify my thinking on several points; and to the students in my classes for Ken Orr and Associates, whose questions and comments helped shape the material presented here.

M. Jean-Dominique Warnier kindly read Chapters 18 and 19, which survey parts of his work, and made several corrections. And Jim Batterson of Best Products provided a detailed critique of an earlier draft of the book, which led to several improvements.

I am grateful to Manufacturers Life for cooperating in my association with KOA, and for giving concrete assistance with this book.

Specific thanks go to Ken McEvoy and Bob Stark, who worked with me in our initial encounter with data structured program design. Ken also advised me on certain aspects of Jackson's work for this book. John Campbell's comments led to the revised treatment of inversion in this edition.

Karen Brown provided meticulous and cheerful editing; but her insistence that data names mustn't change between sentences, and that referenced sections should actually exist, has made reading this book less of an adventure than it might have been.

My father's comments on an early draft had a healthy effect on the book's style.

Finally, I thank my wife, Mary-Anne Sillamaa, for her love and support while this book was being written.

Contents

Part 1

TWICE OVER LIGHTLY

The basic road map of data structured program design is data structure ⟶ process structure ⟶ code.

This Part follows the road twice: once to show that it's familiar territory; then again to study the critical role of Warnier/Orr diagrams.

1

You Already Do It

Even if you've never heard of data structured program design, you regularly use its principles. Here's proof in the form of three simple coding exercises. Please take a couple of minutes now to write routines to the following specifications. (COBOL programmers: Procedure Division code is all that's needed.)

Exercise 1

Display the message:

```
HAPPY
NEW
YEAR
```

Exercise 2

Display either the message:

```
GOOD MORNING
```

or the message:

```
GOOD DAY
```

depending on whether or not the hour is less than 12.

Exercise 3

Display the warning:

```
FLY AT ONCE -- ALL IS DISCOVERED
```

15 times.

If you tried the exercises, your solutions probably look somewhat like the following. For Exercise 1:

```
EXERCISE-1.
      DISPLAY "HAPPY".
      DISPLAY "NEW".
      DISPLAY "YEAR".
```

For Exercise 2:

```
EXERCISE-2.
      IF HOUR < 12
          DISPLAY "GOOD MORNING"
      ELSE
          DISPLAY "GOOD DAY".
```

For Exercise 3:

```
EXERCISE-3A.
      PERFORM DISPLAY-WARNING 15 TIMES.

DISPLAY-WARNING.
      DISPLAY "FLY AT ONCE -- ALL IS DISCOVERED".
```

Or perhaps:

```
EXERCISE-3B.
      MOVE 0 TO WARNING-COUNT.
      PERFORM DISPLAY-WARNING-AND-BUMP-COUNT
          UNTIL WARNING-COUNT = 15.

DISPLAY-WARNING-AND-BUMP-COUNT.
      DISPLAY "FLY AT ONCE -- ALL IS DISCOVERED".
      ADD 1 TO WARNING-COUNT.
```

(If you treated each exercise as a stand-alone program, you would include STOP RUN in each solution. Also, depending on the compiler you use, you might have apostrophes ['] instead of quotation marks ["] around the literals.)

As these exercises demonstrate, you already know the three basic rules of data structured design:

1. When the data is a **sequence** (HAPPY, then NEW, then YEAR), use a simple sequence of instructions.
2. When the data is a **selection** (either GOOD MORNING or GOOD DAY), use a conditional instruction (IF. . .THEN. . .ELSE).
3. When the data is a **repetition** (15 copies of "FLY AT ONCE—ALL IS DISCOVERED"), use a looping instruction (PERFORM. . .TIMES or PERFORM. . .UNTIL).

If you have ever encountered structured programming, you will remember that sequence, selection, and repetition are enough to build any program. But that's like knowing nails and boards are enough to build a house. What you want to know is: Where should you use which?

Data structured design tells you. It says: identify the sequences, selections, and repetitions in your data. Then work out the structure of your process logic by applying the three rules given above.

With this approach you'll be able to produce reports, balance files, navigate data bases, manipulate strings of text, and do calculations. If such matters make up much of your programming load, you'll find data structured design valuable.

It's not the be-all and end-all, though, so we'll also touch on some other useful techniques and point to further reading about them.

2

Data Structure

Chapter 1 was the first trip along the path: data structure ⟶ process structure ⟶ code. In the next three chapters we will follow the path again, stressing the role of the Warnier/Orr diagram.

Warnier/Orr diagrams can represent data structure, process structure, or code. This makes them very useful, not only in handling each phase properly, but also in making the transition from one phase to the next.

This chapter examines the use of Warnier/Orr diagrams in representing data or things. Recall our three messages in the previous chapter; they illustrated the three basic constructs of sequence, selection, and repetition. We'll now see how to diagram each of these constructs, and then look at how to combine them.

2.1 SEQUENTIAL DATA

The first message was:

```
HAPPY
NEW
YEAR
```

We would diagram this like so:

$$\text{MESSAGE 1}\ \begin{cases}\text{"HAPPY"}\\ \text{"NEW"}\\ \text{"YEAR"}\end{cases}$$