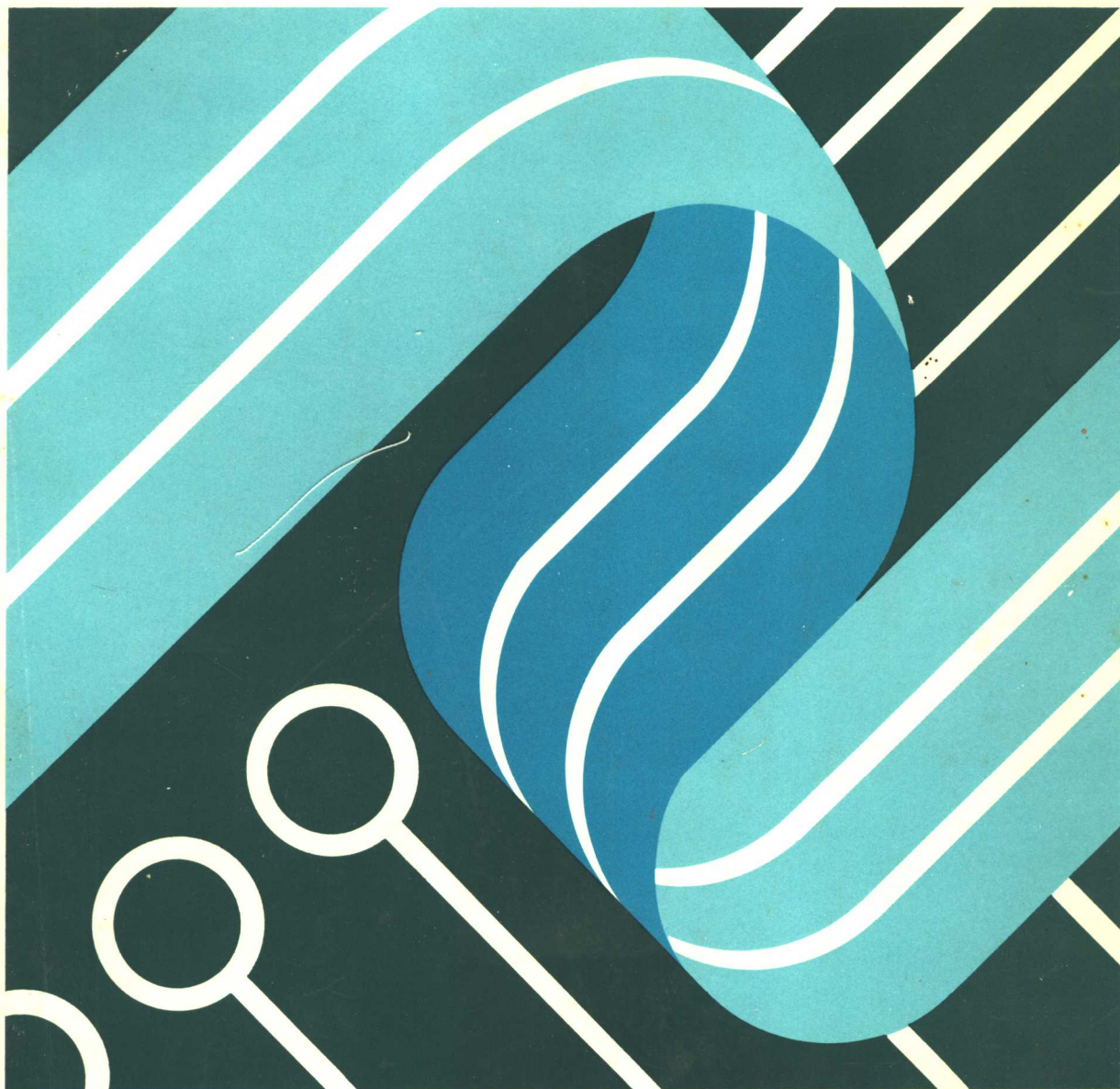


David Marca

Applying Software Engineering Principles



Little, Brown Computer Systems Series

Library of Congress Cataloging in Publication Data

Marca, David.

Applying software engineering principles.

(Little, Brown computer systems series)

1. Electronic digital computers — Programming.

I. Title. II. Series.

QA76.6.M358 1983 001.64'2 83-7964

ISBN 0-316-54574-0

Copyright © 1984 by David Marca

All rights reserved. No part of this book may be reproduced in any form or by any electronic or mechanical means including information storage and retrieval systems without permission in writing from the publisher, except by a reviewer who may quote brief passages in a review.

(Library of Congress Catalog Card No. 83-7964)

ISBN 0-316-54574-0

9 8 7 6 5 4 3 2 1

MV

Published simultaneously in Canada by Little, Brown & Company (Canada) Limited
Printed in the United States of America

Preface

This book is designed for professional programmers and professional-minded students — that is, it is designed for individuals who develop, or intend to develop, software in a professional setting.

Software engineering is the act of an individual who learns to develop software in a practical setting.

Successful software developers seem to have the ability to grow from an existing base of knowledge. They frequently modify their work habits, applying past experiences to new job situations. This book presents software engineering principles that enable programmers to share the experiences of many other professionals.

Sources of Professional Learning

Novice programmers learn about tools and techniques from courses and evaluate their effectiveness by using them on the job. Novices advance to higher levels of professionalism by learning from others. In formal reviews and walkthroughs, they learn ways to apply tools and techniques. From working in small teams, they learn to observe the work habits of others and to evaluate new concepts by placing them in an open forum.

The best professionals learn the difficult and rewarding task of stepping back and observing their own work. Learning comes easier when you know yourself — you can strengthen your abilities, compensate for deficiencies, and identify areas for improvement.

The software engineering principles in this book were developed from all these levels of professional learning. Some were learned from books or schooling, some by watching others work, and some by reflecting on my own work. This is, therefore, a book of experience woven with facts; it is an act of sharing.

Software engineering principles are the *whys* of developing software; they state the reasons behind the programmer's day-to-day work.

Learning from This Book

This book is not overly concerned with syntax, nor with writing FORTRAN statements on coding forms, nor with drawing flowcharts of programs. *This is an*

advanced programming book concerned with the nature of programming once the syntax is learned.

This book covers more than what might normally be considered relevant to an advanced programming course. It describes what you should be *thinking* about when you are programming — that is, it describes how you apply software engineering principles to make a computer language represent a solution to a problem.

The principles and associated examples given throughout the book describe the paradigms I think are the most important ones for the engineering of software. The software examples, written in Microsoft's FORTRAN for the TRS-80 microcomputer, can be implemented on almost any computer.

Software engineering principles generalize the ways of writing software.

These principles and examples will, perhaps, confirm the way you engineer software. Or they may show you a different way of looking at the programming process, describe a new concept for you, or lead you to write down principles of your own. I hope that you will learn as much about the engineering of software from reading this book as I have from writing it.

Why should *you* read this book? Because it will, hopefully, make you think about the way you engineer software. All of the principles and most of the discussions apply to any computer language. So, whether you program exclusively in FORTRAN or in twenty different languages, you should find much related to your work.

David Marca
Research & Development Group
SofTech, Inc.
Waltham, MA
April, 1983

Acknowledgments

The production of a book is never the work of one person. This book is no exception, for it is the synthesis of the knowledge and the experience of many in the computing field. The most appropriate acknowledgment of their work is the collection of references at the end of this book.

I am especially grateful to all the computing professionals I have worked with and have learned from during my career. During the past twelve years, they have openly shared their ideas, listened to my views, and never hesitated to tell me when I was wrong.

Thanks also goes to Jerry Weinberg for his help in directing my writing efforts when no one else could; and to Beth Anderson, developmental editor of the second manuscript, who helped me become a better writer.

Contents

Part I Important Aspects of Software Engineering

3

1	Software Engineering Concepts	4
1.1	Practical Considerations During Software Development	4
1.2	A Need for Engineering Principles	5
1.3	The Software Engineer	5
1.4	Some Engineering Factors	7
	<i>Human Errors</i>	7
	<i>Budget</i>	8
	<i>Performance</i>	10
1.5	Engineering Goals	10
	<i>Timeliness</i>	11
	<i>Efficiency</i>	11
	<i>Reliability</i>	11
	<i>Simplicity</i>	12
	<i>Cost-Effectiveness</i>	12
	<i>Modifiability</i>	13
	<i>Trade-Offs</i>	13
1.6	Keys to Software Engineering	14
	Questions and Exercises	15
	<i>For Further Study</i>	15
	<i>For Your Next Project</i>	15
2	Organizing Engineering Activities	16
2.1	The Software Development Cycle	16
	<i>Analysis</i>	18
	<i>Design</i>	18

	<i>Implementation</i>	18
	<i>Installation</i>	18
2.2	Changing Software	19
2.3	Repetition of the Software Development Cycle	20
2.4	Engineering with the Software Development Cycle	22
2.5	Programming and Programming Products	23
	Questions and Exercises	25
	<i>For Further Study</i>	25
	<i>For Your Next Project</i>	25
3	Fundamentals of Modeling	26
3.1	Models of Software	26
3.2	Model Subject, Purpose, and Viewpoint	26
3.3	Modeling with a Language	28
3.4	Model Form	29
3.5	Creating Models	30
3.6	Validating Models	32
	Questions and Exercises	34
	<i>For Further Study</i>	34
	<i>For Your Next Project</i>	34
4	Engineering with Models	35
4.1	Types of Software Engineering Models	35
	<i>Problem Statement Models</i>	35
	<i>Requirements Models</i>	35
	<i>Design Models</i>	36
	<i>Implementation and Operation Models</i>	36
4.2	Characteristics of Software Engineering Models	36
	<i>Operational Characteristics</i>	36
	<i>Abstractional Characteristics</i>	37
	<i>Language Characteristics</i>	37
4.3	Designs as Models	38
4.4	Programs as Models	40
	Questions and Exercises	41
	<i>For Further Study</i>	41
	<i>For Your Next Project</i>	41

Part II

Engineering with a Computer Language

45

5	Preparing to Code	46
5.1	Learning a Programming Language	46
5.2	Building with Language Primitives	48
5.3	The Template Concept	50
5.4	Compiler Restrictions	51
5.5	Engineering with Templates	52
	Questions and Exercises	54
	<i>For Further Study</i>	54
	<i>For Your Next Project</i>	54
6	Structuring Software	55
6.1	Making Programs	55
6.2	Structuring Data and Algorithms	56
6.3	Limits to Structuring	57
	Questions and Exercises	58
	<i>For Further Study</i>	58
	<i>For Your Next Project</i>	58
7	Structuring Data	59
7.1	FORTRAN Data Types	59
7.2	Arrays	60
7.3	Subscripts	61
7.4	Records	62
7.5	Complex Data Structures	64
	<i>Nodes</i>	64
	<i>Linear Linked Lists</i>	65
	<i>Rings</i>	66
	<i>Trees</i>	66
7.6	Building Data Structures	67
	Questions and Exercises	68
	<i>For Further Study</i>	68
	<i>For Your Next Project</i>	68

8	Structuring the Algorithm	69
8.1	Selection Constructs	69
	<i>IF-THEN Construct</i>	69
	<i>IF-THEN-ELSE Construct</i>	71
	<i>CASE Construct</i>	72
8.2	Repetition Constructs	73
	<i>DO-WHILE Construct</i>	73
	<i>DO Construct</i>	74
	<i>Using Repetition Constructs</i>	75
8.3	Hierarchy and Subprograms	78
	<i>Boundaries</i>	78
	<i>Interface Points</i>	78
	<i>Transfer of Control</i>	78
	<i>Limited Scope of Variables</i>	79
	<i>Representing Subprograms</i>	79
	Questions and Exercises	87
	<i>For Further Study</i>	87
	<i>For Your Next Project</i>	87
9	Structuring Programs	88
9.1	Parallel Structure for Data and Algorithms	88
	<i>Sequencing</i>	88
	<i>Selecting</i>	90
	<i>Repeating</i>	91
	<i>Ordering Hierarchically</i>	93
9.2	Copying and Renaming Data	93
9.3	Manipulating Static Data Structures	94
	<i>Building Static Data Structures</i>	94
	<i>Searching Static Data Structures</i>	96
9.4	Manipulating Dynamic Data Structures	97
	<i>Searching Forward</i>	97
	<i>Searching Backward</i>	98
	<i>Building a Complex Data Structure</i>	100
9.5	Data Reorganization and Program Complexity	101
	<i>Early Reorganization</i>	101
	<i>Late Reorganization</i>	101
	Questions and Exercises	104
	<i>For Further Study</i>	104
	<i>For Your Next Project</i>	105

Part III

Engineering with Existing Software

107

10 The Role of Subprograms in Software Engineering 108

- 10.1 Engineering with Subprograms 108
- 10.2 Isolating and Separating Code into Subprograms 109
 - Summarizing Similarities* 110
 - Summarizing Differences* 110
- 10.3 Reusing and Replacing Subprograms 110
- 10.4 Modularizing to Increase Portability 111
- 10.5 Testing Subprograms 112
 - Phased Testing* 113
 - Incremental Testing* 113
- 10.6 Integrating Subprograms 113
 - Top-Down Integration* 114
 - Bottom-Up Integration* 114
- Questions and Exercises 115
 - For Further Study* 115
 - For Your Next Project* 116

11 Making Good Subprograms 117

- 11.1 Choosing the Correct Kind of Subprogram 117
- 11.2 Supporting Documentation 123
- 11.3 Subprogram Content 127
- 11.4 Coding the Interface 128
- 11.5 Data Transfer Across the Interface 131
- Questions and Exercises 132
 - For Further Study* 132
 - For Your Next Project* 132

12 Software Tools 133

- 12.1 Software Building Blocks 133
- 12.2 The Software Tool Concept 134
- 12.3 Software Tool Properties 134
 - Adaptability* 134
 - Simple Interfaces* 135
 - Simple Functionality* 136

12.4	Types of Software Tools	137
	<i>Subprograms</i>	137
	<i>Subsystems</i>	137
	<i>Programs</i>	138
12.5	Cost Considerations	138
12.6	Using Software Tools	139
	<i>Tool Quality</i>	139
	<i>Tool Applicability</i>	140
	<i>Design Impact</i>	140
	Questions and Exercises	140
	<i>For Further Study</i>	140
	<i>For Your Next Project</i>	141
13	Some Software Development Tools	142
13.1	Source-Level Debugging	142
13.2	Workings of a Source-Level Debugger	143
13.3	Trace and Display Tools	143
13.4	Extending a Computer Language	150
13.5	A FORTRAN String-Handling Facility	152
	<i>Requirements</i>	152
	<i>Defining a Subsystem</i>	152
	<i>Interface Design</i>	153
	<i>Using the Tool</i>	153
	<i>Maintenance Issues</i>	155
	Questions and Exercises	156
	<i>For Further Study</i>	156
	<i>For Your Next Project</i>	156
14	Engineering with Filters	157
14.1	Filter Concepts	157
14.2	Operations of a Filter	159
14.3	Data Streams	160
14.4	Data Transformation Rules	162
	Questions and Exercises	163
	<i>For Further Study</i>	163
	<i>For Your Next Project</i>	164
15	Table-Driven Software	165
15.1	Separating Control and Transformation Logic	165
15.2	Modeling the Behavior of a System	167

15.3 Modeling Software Behavior	171
15.4 Specifying Program Control Rules	177
15.5 Finite-State Machines	181
Questions and Exercises	183
<i>For Further Study</i>	183
<i>For Your Next Project</i>	184

Part IV

Engineering by Separating Concerns

187

16 Human Factors	188
16.1 How a Person Thinks	188
16.2 The Environment Presented by the Computer	189
16.3 Relationships Between the Person and the Computer	190
16.4 Human Limitations	191
<i>Short-Term Memory</i>	191
<i>Channel Capacity</i>	192
16.5 Concepts of Man-Machine Communications	193
16.6 Problem Solving at the Terminal	195
<i>Part Versus Whole</i>	195
<i>Massed Versus Spaced</i>	196
<i>Insight Versus Trial and Error</i>	196
<i>Behavior at the Terminal</i>	196
16.7 Closures	198
Questions and Exercises	199
<i>For Further Study</i>	199
<i>For Your Next Project</i>	200
17 Building User Interfaces	201
17.1 Display Screen Simplification	201
<i>Encoding Data</i>	201
<i>Screen Size</i>	203
<i>Object Compatibility</i>	209
17.2 Designing Within Human Limitations	210
17.3 Types of Dialogues	213
17.4 Some Data Entry Techniques	213

17.5	Increasing User Interface Complexity	216
17.6	Handling Errors	217
	Questions and Exercises	218
	<i>For Further Study</i>	218
	<i>For Your Next Project</i>	218
18	Machine Factors	220
18.1	Main Storage Characteristics	220
18.2	Secondary Storage Characteristics	221
18.3	Compensating for Memory Deficiencies	222
18.4	Impacts on Programming	222
	Questions and Exercises	224
	<i>For Further Study</i>	224
	<i>For Your Next Project</i>	224
19	Building Data Management Software	225
19.1	Physical and Logical Data Management	225
19.2	Separating Data Management Logic	226
19.3	Language Extension for Logical Data Management	227
19.4	Separating Data and Structure	228
	Questions and Exercises	230
	<i>For Further Study</i>	230
	<i>For Your Next Project</i>	230
	Appendix:	
	An Implementation of the	
	“Get Character” Finite-State Machine	233
	References	251
	Index	256

David Marca

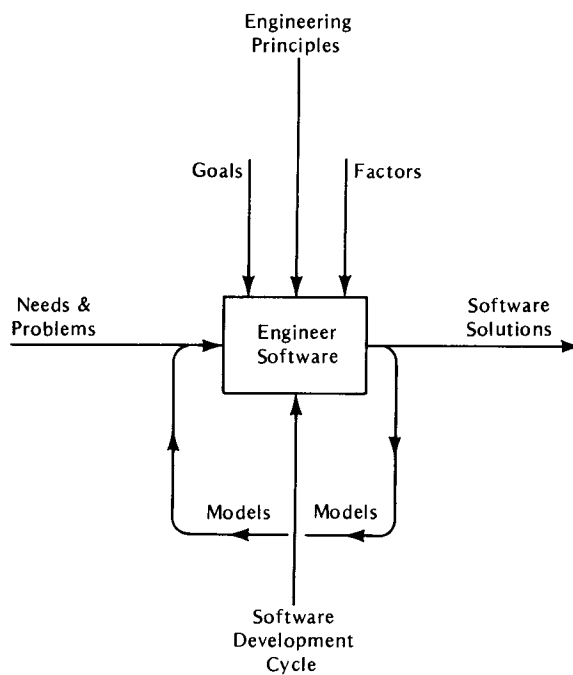
Applying Software Engineering Principles

A GIFT OF
THE ASIA FOUNDATION
BOOKS FOR ASIA
SAN FRANCISCO, CALIFORNIA, U.S.A.
美國亞洲基金會敬贈



Little, Brown and Company
Boston Toronto

FIGURE I-1: A Model of Engineering Software



Part I

Important Aspects of Software Engineering

It is more important to understand about engineering than about coding. That is, software engineers are engineers first and builders of software second; they apply engineering concepts to their work.

Much work precedes the construction of software, though. Software engineers must establish the context for their work. And they must spend some time setting up their projects correctly.

Starting a project is hard, because you must

- define goals,
- formulate an approach to reach those goals, and
- assemble a project team to implement the approach.

To explain, making trade-offs among conflicting factors is a large part of setting goals. Also, defining cycles of software development and developing modeling guidelines help establish an approach. Finally, selecting each individual in the project team requires matching skills and personalities with the approach.

Mistakes made early in a project can jeopardize its success. So software engineers invest time in getting a good start, knowing that the success of the project may depend upon their first few decisions.

PRINCIPLE: Project Start-up

The hardest part of many projects is getting them started correctly.

1 Software Engineering Concepts

Engineering means reaching established goals, within the constraints of the real world, according to a plan. Software is the result of an engineering process in which developers use engineering concepts to build programs. These concepts can be summarized by a set of principles that can be deduced by starting with a very simple fact: Software is engineered in a practical setting [WGU82].

1.1 Practical Considerations During Software Development

Probably the most difficult aspects of building well-engineered software are identifying and assessing practical issues. Software engineers understand that they are performing an engineering activity. What separates software engineers from other programmers is their ability to make decisions with the practical issues in mind during all phases of software development.

There is a distinction between software science and software engineering. The distinction is that the results of engineering need only be useful, not perfect. Getting software to work well enough to be useful is the essence of software engineering [YE79]. But often something very useful cannot be built by proven methods. Software systems fall into this category. So software developers frequently must build systems without an exact science to back them up.

DEFINITION: *Software Engineering*
Software engineering builds useful software.

Today's software systems are considered products of engineering because they cannot be proven to operate correctly, yet they perform very useful functions. In other words, a software system need not be perfect to be useful. For example, the OS/360 project [BF75] comprised hundred of people who developed more than a million lines of code. During initial development of the software no one proved that each module operated correctly. Instead, test cases were used to verify that the operating system