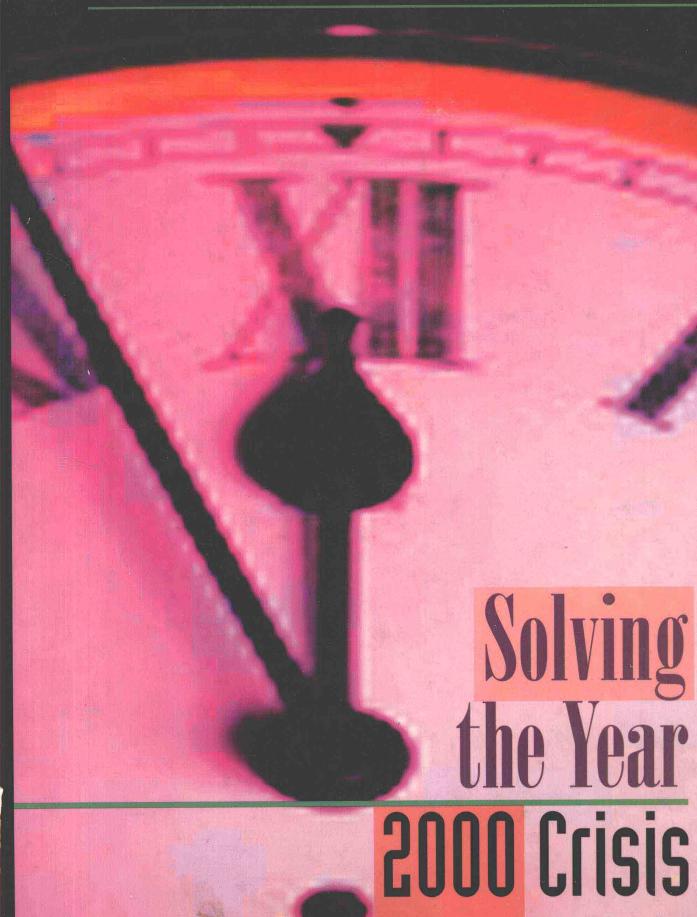
PATRICK MCDERMOTT



# Solving the Year 2000 Crisis

Patrick McDermott



#### Library of Congress Cataloging-in-Publication Data

McDermott, Patrick.

Solving the Year 2000 crisis / Patrick McDermott.

cm.— (Artech House computer science library) Includes bibliographical references and index.

ISBN 0-89006-725-2 (alk. paper)

- 1. Year 2000 date conversion (Computer systems) Title. Ĭ.
- II. Series

OA76.76.S64M37

1998 005.1'6-dc21

98-2923

CIP

#### **British Library Cataloguing in Publication Data**

McDermott, Patrick

Solving the year 2000 crisis.—(Artech House computer science library)

- 1. Year 2000 date conversion (Computer systems)
- I. Title

005.1'6

ISBN 0-89006-725-2

#### Cover and text design by Darrell Judd

© 1998 ARTECH HOUSE, INC. **685 Canton Street** Norwood, MA 02062

All rights reserved. Printed and bound in the United States of America. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from the publisher.

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Artech House cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

International Standard Book Number: 0-89006-725-2 Library of Congress Catalog Card Number: 98-2923

10987654321





# Foreword

VER THE YEARS, Patrick's path and mine have crossed on the winding road to creating software quality. He has taken many of my seminars, learning the various methods and models that advocate the motto of Logical Conclusions, Inc.: "No Bugs."

From the name we have given them, I think of bugs as "innocent" little things that crawl into a system when the original computer programmer isn't looking. It is ironic that my mother, who has never seen a line of computer code in her life, knows what a bug is. What an awful situation the systems profession has gotten itself into when the layperson has a well-known word for such an unprofessional aspect of a business.

Patrick has dedicated a good part of his recent professional life to perhaps the biggest bug known to the layperson. The Y2K problem definitely is not something that accidentally crawled into systems but something that was deliberately set and discovered about as far into systems' life cycles as a bug can get.

The Y2K problem reminds me of a quote I often used in my programming seminars. Edsgar Dykstra, one of the granddaddies of structured programming, said (as near as I can remember), "More programming sins have been committed under the guise of efficiency than any other reason, including blind stupidity."

The Y2K problem stems from an efficiency issue. Ironically, the effort to clear up the problem appears to be costing most companies far more than the original systems' development effort and the savings in storage space over the years.

If only we would learn from such messes and resolve to do better in the future. The way is clear; it only requires following the discipline already available to do it right the first time. Meanwhile, I believe Patrick's book should be of great help in eradicating this especially nasty bug before it metamorphoses into a disastrous epidemic.

Brian Dickinson Logical Conclusions, Inc. Lake Tahoe, California November 1997



# Preface

HEN I WAS IN THE FIFTH GRADE, I lived at Sewart Air Force Base near Smyrna, Tennessee. I delighted in observing the many varieties of insects that inhabited the streams and fields near my home and decided I would be an entomologist when I grew up. That was a momentous decision, since it required abandoning my long-held (since early in the fourth grade) plan to be a Nobel prize-winning chemist. Then I read that chemical companies were one of the largest employers of entomologists, and my two passions were complementary: I would be an entomologist for a chemical company. It was some months before the awful realization struck me that the tie between chemistry and entomology is insecticides and that the goal of most entomologists is to kill as many insects as possible. As a consequence, I abandoned my plans for both chemistry and entomology and eventually found a career that involved exterminating bugs without harming any insects: computer programming. So it would seem natural that I would be drawn to what has been called "the mother of all computer bugs" [1]. The problem of how to deal with

the year 2000, or Y2K, in computer software may seem deceptively simple, but writing this book has drawn on all my quarter century of experience as an economist, a programmer, an analyst, and a manager.

My audience is the workers, both business and technical, and frontline managers actually dealing with the nitty-gritty of the problem in large corporations and government agencies. A number of excellent books are out there that cover planning and initiating Y2K projects, but this book is intended as a view from the trenches covering the practical aspects of actually getting the systems fixed. Perhaps because my background is both computer science and economics, I take a pragmatic approach. The purpose of the book is threefold: (1) to give you the *knowledge* you need to carry out the job, (2) to give you the *sense of urgency* you need to keep going through a tough project, and (3) to impart the *confidence* you need to tackle it.

First, the knowledge. This book is divided into five parts designed to provide Y2K teams with the facts needed to solve the problem. Part I describes the problem in detail; Part II covers the mechanical solutions available to fix the problem; Part III discusses the people issues surrounding the project; Part IV explains how to organize the project; and Part V takes a look at the business aspects of the problem.

Second, I want to give you a sense of urgency. If you have not started your Y2K project, you must do so *now*. As the following table shows, time is running out rapidly. The table shows the number of calendar days, normal workdays, and weekends to January 1, 2000. For many Y2K teams, days and workdays will be the same, since they'll be working through the weekends!

As of	Days	Workdays	Weekends
1/1/98	730	501	104
2/1/98	699	480	99
3/1/98	671	461	95
4/1/98	640	439	91
5/1/98	610	418	87
6/1/98	579	398	82
8/1/98	518	354	74
7/1/98	549	376	78

Countdown to January 1, 2000

As of	Days	Workdays	Weekends
9/1/98	487	333	69
10/1/98	457	312	65
11/1/98	426	291	60
12/1/98	396	272	56
1/1/99	365	251	52
2/1/99	334	231	47
3/1/99	306	212	43
4/1/99	275	189	39
5/1/99	245	167	35
6/1/99	214	147	30
7/1/99	184	125	26
8/1/99	153	10 <del>4</del>	21
9/1/99	122	82	17
10/1/99	92	61	13
11/1/99	61	41	8
12/1/99	31	21	4

Countdown to January 1, 2000 (continued)

Last but not least, I hope to impart confidence. Fixing your Y2K bugs will be a major project and one that will have a planned return on investment (ROI) of zero (some of your projects may have had no ROI, but this is the first time it has been planned that way). But the sky is not going to fall, and with some planning, foresight, common sense, and plain hard work, we'll get through this crisis, although not without some problems. If managed properly, the cost of the project should be in the same order of magnitude as many major development efforts your organization has successfully completed in the past.

The following table lists some possible assessments of the Y2K problem and its eventual effects on the world economy. My assessment is about a 5. I think this is a tough job that will take hard work, but one I am confident we can handle if we get serious about it.

#### Let's just do it!

Score	Assessment		
0	It is a hoax perpetrated by vendors, consultants, and lawyers to make a lot of money.		
1	It is a minor problem that easily can be fixed.		
2	It is no big deal.		
3	The problem should be addressed, but it is not serious.		
4	It is a serious problem if not fixed, but it is well within the realm of what can be done.		
5	It is a serious problem that will require hard work, but it can be fixed without disaster.		
6	The problem can be fixed but will be very expensive.		
7	It is a serious problem that will have huge costs and cripple some large companies.		
8	The costs will be so severe as to cause major economic repercussions.		
9	The problem will cause severe disruptions to our way of life for at least several months.		
10	The problem will be the end of civilization as we know it, and we never will recover.		

How bad is it?

Patrick McDermott Oakland, California May 1998

#### REFERENCE

1. Keogh, J., Solving the Year 2000 Problem, Boston: AP Professional, 1997, p. 7.



# Acknowledgments

NE OF THE NICEST THINGS about the Y2K problem is that I keep running into people I haven't seen in years. Thanks to two friends from the "old days" at the California Division of Labor Statistics for their ideas and encouragement both then and now, Jerry Freeman and Colt Rymer.

Brian Dickinson of Logical Conclusions, Inc., was a source of advice and inspiration, both in reviewing material for this book and in classes I have taken from him over the years. Brian's ideas and teaching have improved the quality of systems developed for the State of California and American President Lines, so I know there is an approach to system development that leads to a logical conclusion.

Y2K also has introduced me to many interesting experiences and acquaintances and some new friends. Thanks to Anthony M. Peeters of San Francisco Computer Consultants, Inc. He has been a source of insight and wisdom, and his selfless work in forming and running the San

Francisco Bay Year 2000 User Group is helping to alleviate the impact of Y2K in the Bay area.

Thanks to the staff and students at the University of California extensions at UC Santa Cruz, UC Davis, UCLA, and UC Berkeley for their help, questions, and comments, which have improved the book tremendously. Special thanks to Nancy Bruss at UC Davis for suggesting a general seminar on the subject that got me started teaching at UC; to Dan Clarke and Sandra Clark for their ideas and support on the Y2K COBOL course at UC Santa Cruz; and to Sandra for her help with turning the concept into a reality.

Bill Payson of Senior Staff 2000 originally suggested the concept of a Y2K COBOL course at UC Santa Cruz that led to my involvement in a project that helped shape this book. His spirit of determination no doubt will continue to inspire us to tackle the Year 2000 challenge.

Mike Gee was born in Okinawa, Japan, where I went to high school. The Japanese word *omoshiroi* means both "fun" and "interesting." It was both fun and interesting being featured in USoft's Year 2000 seminar series in the San Francisco Bay area. Thanks to Mike for including me and for his insights on replacement as a solution to Y2K.

Thanks to Jonathan Plant at Artech House for acquiring the book and his many acts of kindness along the way. It has been a pleasure to work with him and the other fine people at Artech House.

Many thanks to Peter de Jager and the subscribers of his Maillist for many interesting discussions and observations that were both informative and amusing and the spirit of helping others that he has championed.

Words can never adequately thank Lilian Roberts for her support and understanding. Without her encouragement, I might never have completed the project. Her help in crafting sentences about topics she didn't understand and acting interested in what is to most an inherently uninteresting topic significantly contributed to any merit this book might have.



### Contents

Foreword xiii

Preface xv

Reference xviii

#### Acknowledgments xix

#### Part 1

The problem 1
Governments' role in fixing the problem 2

#### Chapter 1

The seven warning signs of Y2K 5 Symptom 1: 2001 > 1999, but 01 < 99 6 Symptom 2: 2001 – 1999 = 2, but 01 – 99 = –98 7 Symptom 3: 00, 99 = Never 8 Symptom 4: The last shall sort first 9 Symptom 5: An interface built for two 10 Symptom 6: The days of our weeks 11 Symptom 7: The one you forgot about 12 When does the 21st century actually begin? 12 The fail-safe principle 13 Embedded systems 14 Other date-range limits 150

Levels of software 15 Part 2 References 16 Solutions 47 Chapter 2 Chapter 5 The good news Replace The very magnitude of the problem When to replace 50 is an ally 18 When *not* to replace 50 Computers are not so smart 19 Advantages, disadvantages, variations 51 Hits before "doomsday" 20 References 51 Scalability 21 Every cloud has a silver(plated) lining 21 Chapter 6 References 23 Expand 53 Converting the data 54 Chapter 3 "Hidden" copies of data 58 The bad news Machine time for conversion 58 What is so hard? 26 Finding the space 59 Recompiling 28 The international standard 60 May the source be with you 29 Bridges and interfaces 60 Other obstacles 30 Logic changes 61 Interdependencies 30 Other calendars 62 Self-fulfilling prophecies 31 "Smart" keys are dumb 63 Some potential disruptions 32 Advantages and disadvantages 64 Getting a straight answer 32 Variations 65 The leap-year algorithm 33 References 65 Birthdays 37 The next Y2K? 38 Chapter 7 The worst that could happen 38 Window 67 References 39 We do windows 67 Fixed window 68 Chapter 4 Sliding window It's not that bad... Pivot year 70 The most expensive problem in history? 41 Objections to windowing 71 Massive bankruptcies and recession? 42 The century domain problem 72 Already too late? 43 I/O or inside? Catastrophe scenarios Standard date library 73 Be prepared! 44 Intercept logic 73 References 45 Advantages and disadvantages 73

Variations 74 Chapter 11 Reference 75 Abandon How bad is it? 101 Chapter 8 Rounds 102 Compress 77 Advantages and disadvantages 103 Digits versus bytes 77 Variations 104 Original formats 80 Compressed formats 81 Chapter 12 **Fulian dates** 82 Code archaeology 105 Lilian 2000 84 The data definitions 106 ABCs 85 What's in a name? 107 Advantages and disadvantages 86 Strange names 107 Variations 87 Misses and false positives 111 References 87 The logic statements 111 Chapter 9 Programmer's comments 112 The data 112 Work around Documentation 114 A provocative (heretical) question 90 The execution path spectrum 114 Is it really broke? 91 System interface 114 Mirabile dictu 91 Hidden code 115 When to *not* fix it 92 The fix might break it 93 Chapter 13 Ride it out 94 Selecting a solution 117 Examples of workarounds 95 Cost factors 118 Advantages and disadvantages of Cost of the fix 118 workarounds 96 Cost to the users 119 Variations 96 Risk 119 Reference 96 Maintainability 120 Performance 121 Chapter 10 Limit 121 Encapsulate Tasks 122 When to encapsulate 98 Restructure? 122 Time warp 98 Convert database? 122 Advantages and disadvantages 99 Fix logic? 123 Variations 100 Recompile? 123 Reference 100 Simultaneity 123 Expansion versus windowing 124 Expansion versus compression 125 Case-by-case approach 126 Use of system date 126 Mottoes and aphorisms 127

#### Part III

The people 129

#### Chapter 14

Staff: The people on your project 131

The labor market 132

Help wanted 133

Rates 136

Factors that affect rates 138

Get some people 140

The coal mines 141

Teaching the basics 142

Loyalty 143

Nonprogramming staff 144

References 144

#### Chapter 15

Consultants 145

Outsource or keep in-house? 146
Code factories 147
Offshore consultants 147
Choosing your consultant 150
References 150

#### Chapter 16

Tools 151

A fool with a tool is still a fool 151

Guarantees 152

Beware analysis paralysis 153

General or language-specific tools? 154

Existing non-Y2K products 155

Y2K compliance 155

Download to PC/LAN? 156

The tool yendors' dilemma 157

Tool classification 158

Non-Y2K categories 158

General categories 159

Y2K-specific tools 163

Reference 163

#### Chapter 17

Estimating 165

Assessment 166
Estimating techniques 166
LOC method 168
Per-program method 169
Per-date method 169
Activity-based method 169
Percentage-of-staff method 170
Large-project method 170
Mutatis mutandi 170
Reference 171

#### Part IV

The project 173

#### Chapter 18

References 197

Strategy 175

Area 1: Scope within the organization 181
Area 2: Scope outside the organization 183
Area 3: Exposure 185
Area 4: Support 186
Area 5: Obstacles 187
Area 6: Control 188
Area 7: Approach 190
Area 8: The seven solutions 191
Area 9: Hedge hopping 192
Area 10: Organization 193
Area 11: Tools 195
Area 12: Vendors 196
Area 13: Staffing 196

#### Chapter 19

Triage 199

Triage groupings 200
Y2K triage levels 201
Don't be greedy 202
Case-by-case decisions 203
References 203

#### Chapter 20

"Fix-it" factories 205

#### Chapter 21

Project planning 209

Phase 1: Assessment 210

Phase 2: Strategy 219

Phase 3: Repair 222 Phase 4: Test 224

Phase 5: Implement 224

Phase 6: Postimplementation 225

Doomsday weekend 225

Doomsday itself 226

References 227

#### Part V

The business 229

#### Chapter 22

The business perspective 231

The plight of the small business 231

Mainframes and desktops 232

Mainframe systems 233

Desktop systems 233

The IS/business line 234

Use your CBBUs 235

Cost in perspective 236

Contingency planning 236

References 237

#### Chapter 23

PCs and desktops 239

It's a mess 240

Rollover test 240

PC fix-its 242

Replacement PCs 242

Spreadsheets 243

The software pyramid 244

Reference 245

#### Chapter 24

Failure points 247

Where to look 247

How to search 248

The seven warning signs 248

Business events 249

A day in the life 250

Criticality 250

Some examples 251

Accounts payable 251

Accounts receivable 252

Licenses and policies 253

Eligibility dates 253

Assumed 19 253

Illogical negative numbers 254

Reports 255

References 255

#### Chapter 25

Testing 257

General testing 258

Who does what? 260

The principles of Y2K testing 260

Cost is independent of consequences 260

Define the results 261

Have someone else test 261

Inspect the results 262

Test for the invalid as well as the valid 262