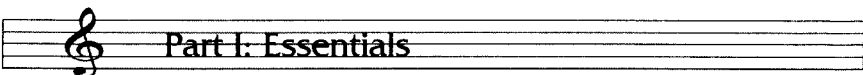


CONTENTS

PREFACE

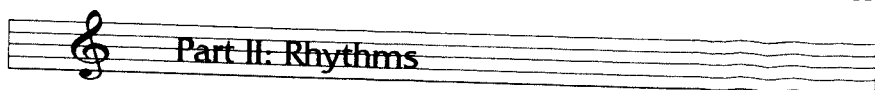
xiii



1	NO PROGRAMMER DIES	3
1.1	Developing Software versus Building a Tunnel	4
1.1.1	The Good Old Days?	5
1.1.2	The More Things Change, the More They Stay the Same?	6
1.1.3	Behind Software Products	7
1.1.4	Deal or No Deal	10
1.2	Do-Re-Mi Do-Re-Mi	12
1.2.1	Iterative Models	14
1.2.2	Code and Fix	16
1.2.3	Chaos	17
1.2.4	Methodology that Matters	21
1.3	Software Development Rhythms	24
1.3.1	Stave Chart by Example	25
1.3.2	Game Theory	28
1.3.3	In-Out Diagram	30
1.3.4	Master-Coach Diagram	31
1.3.5	No Mathematics	32
1.3.6	Where to Explore Rhythms	33
	References	34
2	UNDERSTANDING PROGRAMMERS	37
2.1	Personality and Intelligence	39
2.1.1	Virtuosi	40

2.1.2 Meeting Your Team	41
2.1.3 Recruiting Programmers	43
2.2 Outsourced Programmers	45
2.2.1 Programmers in Their Environments	46
2.2.2 Programmers, Cultures, and Teams	47
2.3 Experienced Management	48
2.3.1 Being Casual about Causal Relationships	49
2.3.2 Not Learning from Experience	50
2.3.3 Doing Things Right <i>Right</i> Now	52
References	54

3 START WITH OPEN SOURCE	55
3.1 Process and Practice	58
3.1.1 The Four Ps of Projects	60
3.1.2 Agile Values	63
3.1.3 Zero-Point Collaboration	64
3.2 Open-Source Software (OSS) Development	65
3.2.1 Software Cloning	66
3.2.2 Software Quality	67
3.2.3 Starting Processes	68
3.2.4 Open-Source Development Community	69
3.2.5 Ugrammers	70
3.2.6 Participant Roles	71
3.2.7 Rapid Release	72
3.2.8 Blackbox Programming	74
3.2.9 OSS Practices	76
3.3 OSS-Like Development	77
3.3.1 Agile Practices	78
3.3.2 Communication Proximity	79
3.3.3 Loose and Tight Couples	80
3.3.4 Collocated Software Development	81
3.4 Conclusion	82
References	83



4 PLAGIARISM PROGRAMMING	87
4.1 Plagiarism	89
4.1.1 Existing Code	90

4.1.2 Social Network Analysis	91
4.1.3 Being Plagiarized	92
4.1.4 Turn Everyone into a Programmer	96
4.1.5 Pattern Language	100
4.1.6 Software Team Capability	102
4.1.7 Rough-Cut Design	105
4.1.8 Training Is Not a Solution	107
4.2 Nothing Faster than Plagiarism	107
4.2.1 Immorality	108
4.2.2 Unprecedented Code	110
4.2.3 People Network	111
4.2.4 Rhythm for Plagiarism	112
4.2.5 Plagiarism at Work	114
4.3 Business and Rhythm for Plagiarism	117
4.3.1 15-Minute Business Presentation	118
4.3.2 Marketing Research	120
4.3.3 Chatting Robot	121
4.3.4 Old Song, New Singer	125
References	129

5 PAIR PROGRAMMING	131
5.1 Art and Science	132
5.1.1 The Right Partner	133
5.1.2 Noisy Programming	134
5.1.3 Just Training	135
5.1.4 Pay to Watch	135
5.2 Two Worlds	136
5.2.1 Moneyless World	137
5.2.2 Money-Led World	139
5.2.3 Economics	140
5.2.4 Mythical Quality-Time	140
5.2.5 Elapsed Time Accelerated	141
5.2.6 Critical Path Method	142
5.2.7 Why Two, Not Three: The Antigroup Phenomenon	145
5.2.8 Software Requirements Are Puzzles	146
5.3 Programming Task Demands	148
5.3.1 2 and 4 Is 6	148
5.3.2 2 and 4 Is 4	149
5.3.3 2 and 4 Is 3	150
5.3.4 2 and 4 $\geq$ 2	151
5.3.5 2 and 4 is Unknown	152

5.4 Pair Programming Is More than Programming	153
5.4.1 Design by Code	154
5.4.2 Pair Design	156
5.4.3 Rhythmic Pair Programming	158
5.5 Pair Programming Team Coached	161
References	162

6 REPEAT PROGRAMMING

6.1 Controversies in Pair Programming	165
6.1.1 Is Programming a Unique Work?	167
6.1.2 Are Three Minds Better than Two?	168
6.1.3 Unreplicable Experiments	169
6.2 Repeat Programming	170
6.2.1 Variances	175
6.2.2 Principles	176
6.2.3 Triple Programming Unproductive	177
6.3 Rhythm: Pair-Solo-Pair-Solo	179
6.3.1 Persistence	179
6.3.2 Connection	181
6.3.3 Motivation	185
6.4 An Exception that Proves Brooks' Law	188
6.4.1 Low Morale	190
6.4.2 Communication Costs	191
6.4.3 Rhythm for Late Projects	192
References	195

7 AGILE TEAMING

7.1 Project Teams	197
7.1.1 Self-Organizing Teams	200
7.1.2 Teams in a Team	202
7.1.3 Project Team Composition	203
7.1.4 Team Lifecycle versus Learning Curve	205
7.2 Productivity	206
7.2.1 The Illusion of Productivity	209
7.2.2 Collective Code Ownership	210
7.2.3 Accountability, Responsibility, and Transparency	210
7.3 Problems and Problem Owners	211
7.3.1 Rhythm: Trouble-Restructuring	212
7.3.2 Teaming Principles	213
	215

7.4	Failing Projects Rescued	217
7.4.1	Project Traffic Light Reporting	219
7.4.2	A Business Case	220
7.4.3	Steering Committee Meeting	220
7.4.4	Agile Teaming in Action	222
7.5	Beware of Iago	222
	References	224
8	INCREMENTAL DESIGN	225
8.1	Modeling and Planning	226
8.1.1	Agile Planning	227
8.1.2	Design by Functional Modules	229
8.1.3	Simple Design	231
8.1.4	Total Cost Concept	232
8.2	Rework or Reuse	234
8.2.1	Unpreventable Rework	235
8.2.2	Improvisation	236
8.2.3	Up-Front Design	238
8.3	Just-in-Time Software Development	239
8.3.1	The CMM Rhythm	240
8.3.2	A Factory Tour	243
8.3.3	Walking Worker	244
8.3.4	Just-in-Time Software Development	246
8.3.5	Incremental Design	247
8.4	Requirements Complexity	249
8.4.1	Forgotten Requirements	251
8.4.2	Conflicting Requirements	252
8.4.3	Rapidly Changing Requirements	253
8.4.4	Requirements and Design	254
8.5	Refactoring	254
8.5.1	Refactoring Activities	258
8.5.2	Refactoring by Challenging	259
8.5.3	Refactoring for Design Patterns	261
8.5.4	Making Deliberate Mistakes	263
	References	263
9	TEST-DRIVEN DEVELOPMENT	265
9.1	Reverse Waterfall	268
9.1.1	Design-Code-Test	268

9.1.2 Test-Code-Design	269
9.2 Test-First Programming	269
9.2.1 Testing and Verification	270
9.2.2 Breakpoint Testing	271
9.2.3 Supporting Practices	272
9.3 Rhythm: Test-Code-Refactor	274
9.3.1 Simple Example	275
9.3.2 Automation	277
9.3.3 Revolution in Consciousness!	278
9.3.4 Test Case for Collaboration	281
9.4 Rapid Software Process Improvement	282
9.4.1 Training Program	283
9.4.2 Project Planning	284
9.4.3 Project Tracking	284
9.4.4 Software Quality	286
9.4.5 Software Configuration	286
9.4.6 People Discipline	288
References	288
EPILOGUE: MEDLEY	291
INDEX	295