# CONTENTS

## PREFACE

### Part I: Essentials

### 1 NO PROGRAMMER DIES

1.1 Developing Software versus Building a Tunnel
   1.1.1 The Good Old Days?  
   1.1.2 The More Things Change, the More They Stay the Same?  
   1.1.3 Behind Software Products  
   1.1.4 Deal or No Deal  

1.2 Do-Re-Mi Do-Re-Mi
   1.2.1 Iterative Models  
   1.2.2 Code and Fix  
   1.2.3 Chaos  
   1.2.4 Methodology that Matters  

1.3 Software Development Rhythms
   1.3.1 Stave Chart by Example  
   1.3.2 Game Theory  
   1.3.3 In–Out Diagram  
   1.3.4 Master–Coach Diagram  
   1.3.5 No Mathematics  
   1.3.6 Where to Explore Rhythms  

References

### 2 UNDERSTANDING PROGRAMMERS

2.1 Personality and Intelligence
   2.1.1 Virtuosi
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 Right 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

Part II: Rhythms

4 PLAGIARISM PROGRAMMING 87
4.1 Plagiarism 89
4.1.1 Existing Code 90
7.4 Failing Projects Rescued  
7.4.1 Project Traffic Light Reporting  
7.4.2 A Business Case  
7.4.3 Steering Committee Meeting  
7.4.4 Agile Teaming in Action  
7.5 Beware of Iago  
References

8 INCREMENTAL DESIGN  
8.1 Modeling and Planning  
8.1.1 Agile Planning  
8.1.2 Design by Functional Modules  
8.1.3 Simple Design  
8.1.4 Total Cost Concept  
8.2 Rework or Reuse  
8.2.1 Unpreventable Rework  
8.2.2 Improvisation  
8.2.3 Up-Front Design  
8.3 Just-in-Time Software Development  
8.3.1 The CMM Rhythm  
8.3.2 A Factory Tour  
8.3.3 Walking Worker  
8.3.4 Just-in-Time Software Development  
8.3.5 Incremental Design  
8.4 Requirements Complexity  
8.4.1 Forgotten Requirements  
8.4.2 Conflicting Requirements  
8.4.3 Rapidly Changing Requirements  
8.4.4 Requirements and Design  
8.5 Refactoring  
8.5.1 Refactoring Activities  
8.5.2 Refactoring by Challenging  
8.5.3 Refactoring for Design Patterns  
8.5.4 Making Deliberate Mistakes  
References

9 TEST-DRIVEN DEVELOPMENT  
9.1 Reverse Waterfall  
9.1.1 Design–Code–Test