201 PRINCIPLES OF SOFTWARE DEVELOPMENT

Alan M. Davis
University of Colorado
at Colorado Springs

McGraw-Hill, Inc.
New York  San Francisco  Washington, D.C.  Auckland  Bogotá
Caracas  Lisbon  London  Madrid  Mexico City  Milan
Montreal  New Delhi  San Juan  Singapore
Sydney  Tokyo  Toronto
CONTENTS

Preface xi

Acknowledgments xv

Chapter 1. Introduction 3

Chapter 2. General Principles 7

Principle 1. Quality Is #1 8
Principle 2. Quality Is in the Eyes of the Beholder 9
Principle 3. Productivity and Quality Are Inseparable 10
Principle 4. High-Quality Software Is Possible 11
Principle 5. Don't Try to Retrofit Quality 12
Principle 6. Poor Reliability Is Worse Than Poor Efficiency 13
Principle 7. Give Products to Customers Early 14
Principle 8. Communicate with Customers/Users 15
Principle 9. Align Incentives for Developer and Customer 16
Principle 10. Plan to Throw One Away 17
Principle 11. Build the Right Kind of Prototype 18
Principle 12. Build the Right Features into a Prototype 19
Principle 13. Build Throwaway Prototypes Quickly 20
Chapter 3. Requirements Engineering Principles 47

Principle 38. Poor Requirements Yield Poor Cost Estimates 48
Principle 39. Determine the Problem Before Writing Requirements 49
Principle 40. Determine the Requirements Now 50
Principle 41. Fix Requirements Specification Errors Now 51
Principle 42. Prototypes Reduce Risk in Selecting User Interfaces 52
Principle 43. Record Why Requirements Were Included 53
Principle 44. Identify Subsets 54
Principle 45. Review the Requirements 55
Principle 46. Avoid Design in Requirements 56
Principle 47. Use the Right Techniques 57
Principle 48. Use Multiple Views of Requirements 58
Principle 49. Organize Requirements Sensibly 59
Principle 50. Prioritize Requirements 60
Principle 51. Write Concisely 61
Principle 52. Separately Number Every Requirement 62
Principle 53. Reduce Ambiguity in Requirements 63
Principle 54. Augment, Never Replace, Natural Language 64
Principle 55. Write Natural Language Before a More Formal Model
Principle 56. Keep the Requirements Specification Readable 66
Principle 57. Specify Reliability Specifically 67
Principle 58. Specify When Environment Violates "Acceptable" Behavior 68
Principle 59. Self-Destruct TBDs 69
Principle 60. Store Requirements in a Database 70

Chapter 4. Design Principles 73

Principle 61. Transition from Requirements to Design Is Not Easy 74
Principle 62. Trace Design to Requirements 75
Principle 63. Evaluate Alternatives 76
Principle 64. Design Without Documentation Is Not Design 77
Principle 65. Encapsulate 78
Principle 66. Don't Reinvent the Wheel 79
Principle 67. Keep It Simple 80
Principle 68. Avoid Numerous Special Cases 81
Principle 69. Minimize Intellectual Distance 82
Principle 70. Keep Design Under Intellectual Control 83
Principle 71. Maintain Conceptual Integrity 84
Principle 72. Conceptual Errors Are More Significant Than Syntactic Errors 85
Principle 73. Use Coupling and Cohesion 86
Principle 74. Design for Change 87
Chapter 5. Coding Principles 101

Principle 87. Avoid Tricks 102
Principle 88. Avoid Global Variables 103
Principle 89. Write to Read Top-Down 104
Principle 90. Avoid Side-Effects 105
Principle 91. Use Meaningful Names 106
Principle 92. Write Programs for People First 107
Principle 93. Use Optimal Data Structures 108
Principle 94. Get It Right Before You Make It Faster 109
Principle 95. Comment Before You Finalize Your Code 110
Principle 96. Document Before You Start Coding 111
Principle 97. Hand-Execute Every Component 112
Principle 98. Inspect Code 113
Principle 99. You Can Use Unstructured Languages 114
Principle 100. Structured Code Is Not Necessarily Good Code 115
Principle 101. Don’t Nest Too Deep 116
Principle 102. Use Appropriate Languages 117
Chapter 6. Testing Principles  123

Principle 107. Trace Tests to Requirements  124
Principle 108. Plan Tests Long Before It Is Time to Test  125
Principle 109. Don’t Test Your Own Software  126
Principle 110. Don’t Write Your Own Test Plans  127
Principle 111. Testing Exposes Presence of Flaws  128
Principle 112. Though Copious Errors Guarantee Worthlessness, Zero Errors Says Nothing About the Value of Software  129
Principle 113. A Successful Test Finds an Error  130
Principle 114. Half the Errors Found in 15 Percent of Modules  131
Principle 115. Use Black-Box and White-Box Testing  132
Principle 116. A Test Case Includes Expected Results  133
Principle 117. Test Invalid Inputs  134
Principle 118. Always Stress Test  135
Principle 119. The Big Bang Theory Does Not Apply  136
Principle 120. Use McCabe Complexity Measure  137
Principle 121. Use Effective Test Completion Measures  138
Principle 122. Achieve Effective Test Coverage  139
Principle 123. Don’t Integrate Before Unit Testing  140
Principle 124. Instrument Your Software  141
Principle 125. Analyze Causes for Errors  142
Principle 126. Don’t Take Errors Personally  143

Chapter 7. Management Principles  145

Principle 127. Good Management Is More Important Than Good Technology  146
Principle 128. Use Appropriate Solutions  147
Principle 129. Don’t Believe Everything You Read  148
Principle 130. Understand the Customers’ Priorities  149
Principle 131. People Are the Key to Success  150
Principle 132. A Few Good People Are Better Than Many Less Skilled People  151
Principle 133. Listen to Your People  152
Principle 134. Trust Your People  153
Principle 135. Expect Excellence  154
Principle 136. Communication Skills Are Essential  155
Principle 137. Carry the Water  156
Principle 138. People Are Motivated by Different Things  157
Principle 139. Keep the Office Quiet  158
Principle 140. People and Time Are Not Interchangeable  159
Principle 141. There Are Huge Differences Among Software Engineers  160
Principle 142. You Can Optimize Whatever You Want  161
Principle 143. Collect Data Unobtrusively  162
Principle 144. Cost Per Line of Code Is Not Useful  163
Principle 145. There Is No Perfect Way to Measure Productivity  164
Principle 146. Tailor Cost Estimation Methods  165
Principle 147. Don’t Set Unrealistic Deadlines  166
Principle 148. Avoid the Impossible  167
Principle 149. Know Before You Count  168
Principle 150. Collect Productivity Data  169
Principle 151. Don’t Forget Team Productivity  170
Principle 152. LOC/PM Independent of Language  171
Principle 153. Believe the Schedule  172
Principle 154. A Precision-Crafted Cost Estimate Is Not Foolproof  173
Principle 155. Reassess Schedules Regularly  174
Principle 156. Minor Underestimates Are Not Always Bad  175
Principle 157. Allocate Appropriate Resources  176
Principle 158. Plan a Project in Detail  177
Principle 159. Keep Your Plan Up-to-Date  178
Principle 160. Avoid Standing Waves  179
Chapter 8. Product Assurance Principles  193

Principle 173. Product Assurance Is Not a Luxury  194
Principle 174. Establish SCM Procedures Early  195
Principle 175. Adapt SCM to Software Process  196
Principle 176. Organize SCM to Be Independent of Project Management  197
Principle 177. Rotate People Through Product Assurance  198
Principle 178. Give Every Intermediate Product a Name and Version  199
Principle 179. Control Baselines  200
Principle 180. Save Everything  201
Principle 181. Keep Track of Every Change  202
Principle 182. Don’t Bypass Change Control  203
Principle 183. Rank and Schedule Change Requests  204
Principle 184. Use Validation and Verification (V&V) on Large Developments  205

Chapter 9. Evolution Principles  207

Principle 185. Software Will Continue to Change  208
Principle 186. Software’s Entropy Increases  209
Principle 187. If It Ain’t Broke, Don’t Fix It 210
Principle 188. Fix Problems, Not Symptoms 211
Principle 189. Change Requirements First 212
Principle 190. Prerelease Errors Yield Postrelease Errors 213
Principle 191. The Older a Program, the More Difficult It Is to Maintain 214
Principle 192. Language Affects Maintainability 215
Principle 193. Sometimes It Is Better to Start Over 216
Principle 194. Renovate the Worst First 217
Principle 195. Maintenance Causes More Errors Than Development 218
Principle 196. Regression Test After Every Change 219
Principle 197. Belief That a Change Is Easy Makes It Likely It Will Be Made Incorrectly 220
Principle 198. Structuring Unstructured Code Does Not Necessarily Improve It 221
Principle 199. Use Profiler Before Optimizing 222
Principle 200. Conserve Familiarity 223
Principle 201. The System’s Existence Promotes Evolution 224

References Index 225

Subject Index 235
If software engineering is really an engineering discipline, it is the intelligent application of proven principles, techniques, languages, and tools to the cost-effective creation and maintenance of software that satisfies users' needs. This book is the first collection of software engineering principles ever written in one volume.* A principle is a basic truth, rule, or assumption about software engineering that holds regardless of the technique, tool, or language selected. With few exceptions, the principles published here are not original. They have been extracted from the writings of many software engineering practitioners and researchers. These individuals have been unselfish enough to share their experiences, ideas, and wisdom with all of us. I make no claim that these 201 principles are mutually exclusive. Unlike Boehm's seven "basic" software engineering principles, a combination of some of these principles may imply another. I also make no claim that these 201 principles are 100 percent compatible. The adages, "Absence makes the heart grow fonder" and "Out of sight, out of mind" are each true, and each can be applied to life, but they cannot both be used to justify the same decision. The principles contained in this volume are all valid, and they can all be used to improve software engineering, but it may be impossible to apply some combinations of them on any one project.

*Winston Royce and Barry Boehm published the first two papers on software engineering principles with five and seven principles, respectively [ROY70, BOE83].
Manny Lehman [LEH80] has stated eloquently why principles underlying software engineering are inherently different from principles underlying other areas of human exploration. He states there is no reason to expect such principles to have the same "precision and predictability of [say] the laws of physics." The reason for this is that, unlike physics or biology, the process of software development is "managed and implemented by people; thus in the long term [its behavior should] be expected to be unpredictable, dependent on the judgments, whims, and actions of [people]." On the other hand, software does seem to exhibit many regular and predictable traits [LEH80]. These lead to many basic principles that can be enumerated and used by inexperienced and experienced software engineers and managers to enhance the quality of both the software engineering process and software products.

The purpose of this book is to present in one volume the principles of software engineering as a reference guide. It is aimed at three classes of readers:

1. Software engineers and managers. In this book you can find out what is good and what is not. If you are new at software engineering or software management, here is a place to find out what you need to know.

2. Students of software engineering. For students, there are two primary uses of this book. First, here are the basic, nondogmatic tenets that every software engineer should know. Second, the references in these pages point to some of the best papers and books ever written on software engineering. If you do nothing other than read the items referenced, this book will have been successful, and you will have been exposed to a wealth of knowledge.

3. Software researchers. Researchers may often find it difficult to find the original source of an idea. I have provided references to publications that reflect either the original source or an alternative, excellent work that refers to the original source.

I sincerely hope that everybody who buys this book attempts to read as many of the referenced works as possible. My brief description of the principle is intended to be friendly, easy-to-read, and insightful. But for real appreciation you need to read the referenced works. These works are not neces-
sarily the original source of the idea (although in many cases they are). Nor is the given principle necessarily a primary point of the reference. In every case, however, the referenced work contains a wealth of helpful background, insight, justification, backup data, or information related to the principle.

In summary, this book should be the first place for you to look up any software engineering idea. However, this is a book of principles, not techniques, languages, or tools. You will not find out how to use any techniques, languages, or tools which the principles described here transcend. Furthermore, this book tries to avoid all fads, good or bad! For the most part, fads are popular for three to ten years, then lose favor. The underlying principles that might be behind a fad can be found in this volume, but not the fad itself. Thus, for example, you won't see any reference to object-orientation per se here, but you will find many references to the principles underlying object-orientation, such as encapsulation.

The principles are organized into general categories to aid in finding them and to aid in relating similar principles. These categories correspond to primary phases of software development (that is, requirements, design, etc.) and to other critical "support" activities, such as management, product assurance, and so on, as shown in Fig. P-1.

Alan M. Davis
REFERENCES


ACKNOWLEDGMENTS

Stephen Andriole of Drexel University unknowingly inspired me to write this book during a class we were coteaching. I had just mentioned that software engineering, like all engineering disciplines, is driven by a set of underlying principles. My statement seemed quite logical. However, Steve challenged me: “Name one, Al. Just name one!” Luckily I think well on my feet and came up with one. He said, “Okay, name just one more, and I’ll believe that there really are software engineering principles.” I thought of another, and another.

Kerry Baugh was responsible for creating and maintaining the physical quality of the manuscript. Steve Andriole, Manny Lehman, and Jawed Siddiqi made contributions by reviewing early versions of the manuscript. Dr. Siddiqi was instrumental in providing advice on how to organize the principles.

Last, but not least, I want to thank my wife, Ginny, our children, Marsha and Michael, and my parents, Barney and Hannah Davis, for all their love and support and willingness to live without husband, father, or son while I spent hours writing.