Contents

About the Author .... *xi*

Acknowledgments .... *xiii*

Introduction .... *xv*

Chapter 1 Preventative Medicine *1*

1.1 Core Problems *2*
- 1.1.1 Time to Market Pressure *3*
- 1.1.2 Fluid Specifications *5*
- 1.1.3 Trade Features for Time *6*
- 1.1.4 Get It in Writing *7*
- 1.1.5 Complexity *10*

1.2 Defensive Programming *12*
- 1.2.1 Cohesion and Coupling *12*
- 1.2.2 Checking for Bad Input *16*
- 1.2.3 Data Scope *23*
- 1.2.4 Logging *26*
- 1.2.5 Documentation *36*
- 1.2.6 Design for Change *39*
- 1.2.7 Incremental Refinement *42*

1.3 Unit Testing *43*
- 1.3.1 Motivation Behind Automated Testing *44*
- 1.3.2 Steps Towards a Framework *47*
- 1.3.3 Framework Extensions *53*

1.4 Tool Configuration *57*
- 1.4.1 Use Compiler Warnings *57*
- 1.4.2 Build Settings *59*

1.5 Machine Dependencies *61*
- 1.5.1 Endianess *61*
- 1.5.2 Memory Alignment *62*
- 1.5.3 Data Type Sizes *64*
- 1.5.4 The Benefits of a Virtual Machine *66*

1.6 Summary *67*
- The Bottom Line: Why Do Bugs Occur? *67*
- Refinement Checklist: Proactive Bug Prevention *68*

Chapter 2 Debugging Tactics 71

2.1 Initial Steps 72
2.1.1 Duplicate the Bug 72
2.1.2 Bugs That Can't Be Duplicated 72
2.1.3 Verify the Bug Is Genuine 77
2.2 Resolving the Bug: Quick Fixes 78
2.2.1 Examine Recent Changes 78
2.2.2 Use Tracing Information 79
2.2.3 Déjà Vu 79
2.2.4 Know When to Quit 80
2.3 Resolving the Bug: The Scientific Method 80
2.3.1 General Approach 80
2.3.2 Locating the Bug: Incremental Integration 82
2.3.3 Locating the Bug: Binary Search 83
2.3.4 Understand the Problem 84
2.3.5 Preventing Careless Errors 86
2.3.6 Diagnostic Tools 88
2.3.7 Basic Debugger Operation 98
2.4 Record Keeping 105
2.4.1 Individual Record Keeping 105
2.4.2 Collaborative Record Keeping 106
2.5 Summary 110

Chapter 3 Understand the Problem 113

3.1 How Knowledge Is Lost 114
3.1.1 Competition 115
3.1.2 Attrition 118
3.1.3 Promotion 119
3.2 Poorly Written Code 120
3.2.1 Design Problems 121
3.2.2 Obfuscation 125
3.2.3 Misleading Code 136
3.3 Reverse Engineering 138
3.3.1 General Strategies 138
3.3.2 Countermeasures 146
3.3.3 Creating a Knowledge Base 153
3.4 Summary 156

Chapter 4 Debugger Internals 157

4.1 Types of Debuggers 158
4.1.1 Machine Debuggers vs. Symbolic Debuggers 158
4.1.2 Debugging Infrastructures: Custom Built 165
4.1.3 Debugging Infrastructures: System Calls 178
4.1.4 Debugging Infrastructures: Interpreters 195
4.1.5 Kernel Debuggers 199
4.1.6 Interface: Command Line vs. GUI 202
4.2 Symbolic Debugger Extensions 203
4.2.1 Dynamic Breakpoints 203
4.2.2 Single Stepping 204
4.3 Countertactics 206
4.3.1 System Calls 206
4.3.2 Remove Debug Information 207
4.3.3 Code Salting 209
4.3.4 Mixed Memory Models 210
4.4 Summary 211

Chapter 5 Optimization: Memory Footprint 215

5.1 Forgotten History 217
5.2 Program Layout in Memory 219
5.2.1 Scenario: A Single Segment 220
5.2.2 Scenario: Code and Data Segments Only 222
5.2.3 Scenario: All Four Segment Types 223
5.3 Code Segment 224
5.3.1 Cut-and-Paste Programming 225
5.3.2 Macros 228
5.3.3 Dead Code 230
5.4 Data Segment 231
5.4.1 Dual-Use Data Structures 231
5.4.2 Bit Fields 233
5.4.3 Compression Algorithms 235
5.5 Stack Segment 238
5.5.1 Activation Records 239
5.5.2 Function Parameters 245
5.5.3 Local Variables 247

5.6 Heap 248
5.6.1 Memory Pools 249
5.6.2 Recycling 254
5.6.3 Lazy Instantiation 255
5.6.4 Tracking Memory Usage 258
5.7 Summary 260

Chapter 6 Optimization: CPU Cycles 263

6.1 Program Control Jumps 264
6.1.1 Labels and GOTO 264
6.1.2 Function Parameters 266
6.1.3 Functions with a Varying Number of Arguments 268
6.1.4 System Calls 269
6.1.5 Recursion 274
6.2 Program Control Branching 274
6.2.1 Lookup Tables 275
6.2.2 switch vs. if-else 277
6.2.3 Common Case First, Infrequent Case Last 279
6.3 Program Control Loops 280
6.3.1 Loop Invariants 280
6.3.2 Function Calls 282
6.3.3 Array References 283
6.3.4 Breaking Up Compound Boolean Expressions 285
6.3.5 Loop Unrolling 286
6.3.6 Loop Jamming 286
6.3.7 Extracting Program Branching Statements 287
6.4 Memory Management 288
6.4.1 Dealing with the Overhead 288
6.4.2 Locality of Reference 292
6.5 Input/Output 294
6.5.1 Caching 295
6.5.2 Buffering 296
6.5.3 Advanced Techniques 298
6.6 Exceptions 301
6.6.1 Dynamic Registration Model 304
6.6.2 Static Table Model 305
6.6.3 Dealing with Overhead 306
6.6.4 Abusing Exceptions 308

6.7 Expensive Operations 308
6.7.1 Eliminate Common Subexpressions 308
6.7.2 Floating Point Calculation Myths 309
6.7.3 Strength Reduction 311
6.7.4 Synchronization 311
6.7.5 Shorthand Operator Myths 315
6.8 Quick Fixes 316
6.8.1 Better Hardware 316
6.8.2 Constrain the Problem 316
6.8.3 Compiler Settings 317
6.9 Summary 318
6.10 Putting It All Together 320

Chapter 7 Final Words of Advice 321

7.1 Other Threats to Source Code Integrity 322
Fashionable Technology: A Case Study 323
Brainwashing 101 324
The Real Issue 324
7.2 Maintaining a Paper Trail 325
Quietly Keep Records 325
The Myth of Privacy 326
7.3 History Repeats Itself 327
The "New Economy" Hits Home 328

Index 331