Sumário

Prefácio **11**

A História das Regras **15**

Como discordar das Regras **18**

Regra 1 ■ O mais simples possível, mas não mais simples do que isso **20**

Avaliação da simplicidade 22
... Mas não mais simples do que isso 23
Às vezes é melhor simplificar o problema e não a solução 25
Algoritmos simples 29
Não perca o rumo 31
Uma Regra que regulamenta todas 33

Regra 2 ■ Bugs são contagiosos **34**

Não dependa de seus usuários 35
Os testes automatizados podem ser complicados 36
Um código stateless é mais fácil de testar 37
Audite o estado que não puder eliminar 43
Não confie no chamador 44
Mantenha seu código íntegro 50

Regra 3 ■ Um bom nome é a melhor documentação **51**

Não otimize usando menos pressionamentos de teclas 52
Não misture convenções 54
Não dê um tiro no pé 57
Não me faça pensar 60

Regra 4 ■ A generalização demanda três exemplos **63**

YAGNI 66
Uma objeção óbvia a essa estratégia, em resposta à qual eu dobro a aposta 68
Na verdade, é pior do que o padrão YAGNI 71
Eu não chamaria isso de sucesso 77

Regra 5 ■ A primeira lição para a otimização é não otimizar **79**

Primeira lição da otimização 83
Segunda lição da otimização 83
Teste a segunda lição 83

Etapa 1: Meça e atribua o tempo do processador ... 84
Etapa 2: Verifique se não é um bug ... 84
Etapa 3: Avalie seus dados ... 84
Etapa 4: Planeje e crie um protótipo ... 85
Etapa 5: Otimize e repita ... 85
Aplicação do processo de otimização de cinco etapas ... 86
Não existe terceira lição da otimização ... 91

Interlúdio: no qual o capítulo anterior é criticado ... 92

Regra 6 ■ Revisões de código são boas por três razões ... 96

As revisões de código permitem compartilhar conhecimento ... 99
Revisão de código proibida ... 100
O real valor da revisão de código ... 101
As revisões de código são inerentemente sociais ... 101

Regra 7 ■ Elimine casos de falha ... 103

Uma função que torna fácil dar um tiro no próprio pé ... 104
Atirando no próprio pé via ricochete ... 105
Uso da ajuda do compilador para evitar o tiro no pé ... 107
A hora certa é o que importa ... 108
Um exemplo mais complicado ... 108
Faça com que erros de ordenação não possam ocorrer ... 112
Uso de templates em substituição ao encadeamento de métodos ... 114
Controle de estado coordenado ... 115
É bom detectar erros, mas tornar impossível expressá-los é melhor ... 119

Regra 8 ■ Código que não é executado não funciona ... 120

Etapa 1: Um começo simples ... 121
Etapa 2: Generalização de um padrão comum ... 124
Etapa 3: Inclusão de disfarces ... 125
Etapa 4: O feitiço vira contra o feiticeiro ... 127
Atribuição de culpa ... 128
Limites dos testes ... 129

Regra 9 ■ Escreva código resumível ... 132

É assim que nos sentimos quando não temos sucesso ... 135
Papel da memória de curto prazo ... 135
Onde definir o limite ... 139
Custo da abstração ... 141
Use a abstração para facilitar o entendimento ... 142
Papel da memória de longo prazo ... 142
O conhecimento comum é gratuito; novos conceitos são caros ... 144
Juntando tudo ... 147

Regra 10 ■ Isole a complexidade..... 148
Um exemplo simples 149
Ocultação de detalhes internos..... 150
Estado distribuído e complexidade 152
Capacitado? 155
As coisas começam a ficar nebulosas 157
Repensando a abordagem..... 160
Complexidade isolada, interações simples..... 163

Regra 11 ■ É duas vezes melhor?..... 165
Três caminhos para seguir em frente: ignore, ajuste ou refatore 166
Evolução gradual versus reinvenção contínua..... 167
Princípio básico simples 169
Lidando com benefícios nebulosos 171
Reformular é uma boa oportunidade para corrigir pequenos problemas 171

Regra 12 ■ Equipes grandes precisam de convenções fortes.....173
Convenções de formatação.....174
Convenções de uso da linguagem 175
Convenções de solução de problemas.....176
Membros de equipes eficazes pensam da mesma forma..... 181

Regra 13 ■ Encontre a pedra que começou a avalanche 184
Ciclo de vida de um bug..... 184
Redução do estado..... 188
Como lidar com um estado inevitável..... 193
Como lidar com atrasos inevitáveis..... 196

Regra 14 ■ Existem quatro tipos de código..... 198
Problema Fácil, solução Simples 199
Problema Fácil, três soluções Complicadas 200
Custo da complexidade..... 204
Os quatro (que na verdade são três) tipos de programadores..... 205
Problema Difícil, soluções um pouco Complicadas que não funcionam 206
Problema Difícil, solução um pouco Complicada 208
Problema Difícil, solução Simples 212

Regra 15 ■ Remova as ervas daninhas.....213
Identificação de ervas daninhas 216
Como o código passa a apresentar ervas daninhas 217

Regra 16 ■ Trabalhe para trás a partir de seu resultado e não para a frente a partir do código .. 220
Um exemplo 221
Surge um incômodo 225

Escolha um lado da lacuna 228
Trabalhando para trás 228
E agora algo totalmente diferente 232
Trabalhando para a frente e para trás 240

Regra 17 ■ Às vezes o problema maior é mais fácil de resolver 242

Conclusões apressadas 242
Busca de um caminho claro para seguir 248
Reconhecendo a oportunidade 252

Regra 18 ■ Deixe seu código contar a própria história 254

Não conte histórias que não sejam reais 255
Certifique-se de que a história seja relevante 256
Conte boas histórias 258

Regra 19 ■ Reformule em paralelo 263

Obstáculos no caminho 263
Crie um sistema paralelo 265
Um exemplo concreto 265
Alocação de pilha na prática 268
Uma nuvem no horizonte 271
Torne os contextos de pilha mais inteligentes 272
Migração dos contextos de pilha antigos para os novos 277
Prepare-se para migrar StackVector 279
Hora de migrar 282
Reconhecendo quando a reformulação paralela é uma boa estratégia 283

Regra 20 ■ Faça o cálculo 284

Automatizar ou não automatizar 285
Procure limites rígidos 287
Quando o cálculo muda 291
Quando o problema matemático volta a ser um problema do Word 292

Regra 21 ■ Às vezes só precisamos martelar os pregos 294

Um novo argumento 295
Nunca há apenas um bug 297
A tendência de automatizar 299
Gerencie o tamanho dos arquivos 300
Não existem atalhos 301

Conclusão: Adapte as Regras para você 302

Use o bom senso 303
Façam debates entre vocês 304
Hora de me despedir 305

Apêndice A ■ Leitura de C++ para programadores Python 306
Tipos 306
Formatação e comentários 308
Comentários 310
Indentação e divisão de linhas 310
Operações booleanas 311
Listas 311
Operadores de incremento 312
Classes 312
Visibilidade 316
Declarações e definições 316
Sobrecarga de funções 318
Templates 319
Ponteiros e referências 321

Apêndice B ■ Leitura de C++ para programadores JavaScript 324
Tipos 324
Arrays 326
Classes 327
Declarações e definições 332
Sobrecarga de funções 334
Templates 335
Ponteiros e referências 336

Sobre o autor 339

Índice remissivo 341