

SECOND EDITION

BEGINNING
Software
Engineering

Rod Stephens

WILEY

BEGINNING SOFTWARE ENGINEERING

INTRODUCTION	xxvii
--------------------	-------

► PART I SOFTWARE ENGINEERING STEP-BY-STEP

CHAPTER 1	Software Engineering from 20,000 Feet	3
CHAPTER 2	Before the Beginning	15
CHAPTER 3	The Team	29
CHAPTER 4	Project Management	53
CHAPTER 5	Requirements Gathering	81
CHAPTER 6	High-Level Design	117
CHAPTER 7	Low-Level Design	155
CHAPTER 8	Security Design	185
CHAPTER 9	User Experience Design	209
CHAPTER 10	Programming	245
CHAPTER 11	Algorithms	273
CHAPTER 12	Programming Languages	307
CHAPTER 13	Testing	327
CHAPTER 14	Deployment	359
CHAPTER 15	Metrics	371
CHAPTER 16	Maintenance	401

► PART II PROCESS MODELS

CHAPTER 17	Predictive Models	427
CHAPTER 18	Iterative Models	445
CHAPTER 19	RAD	465

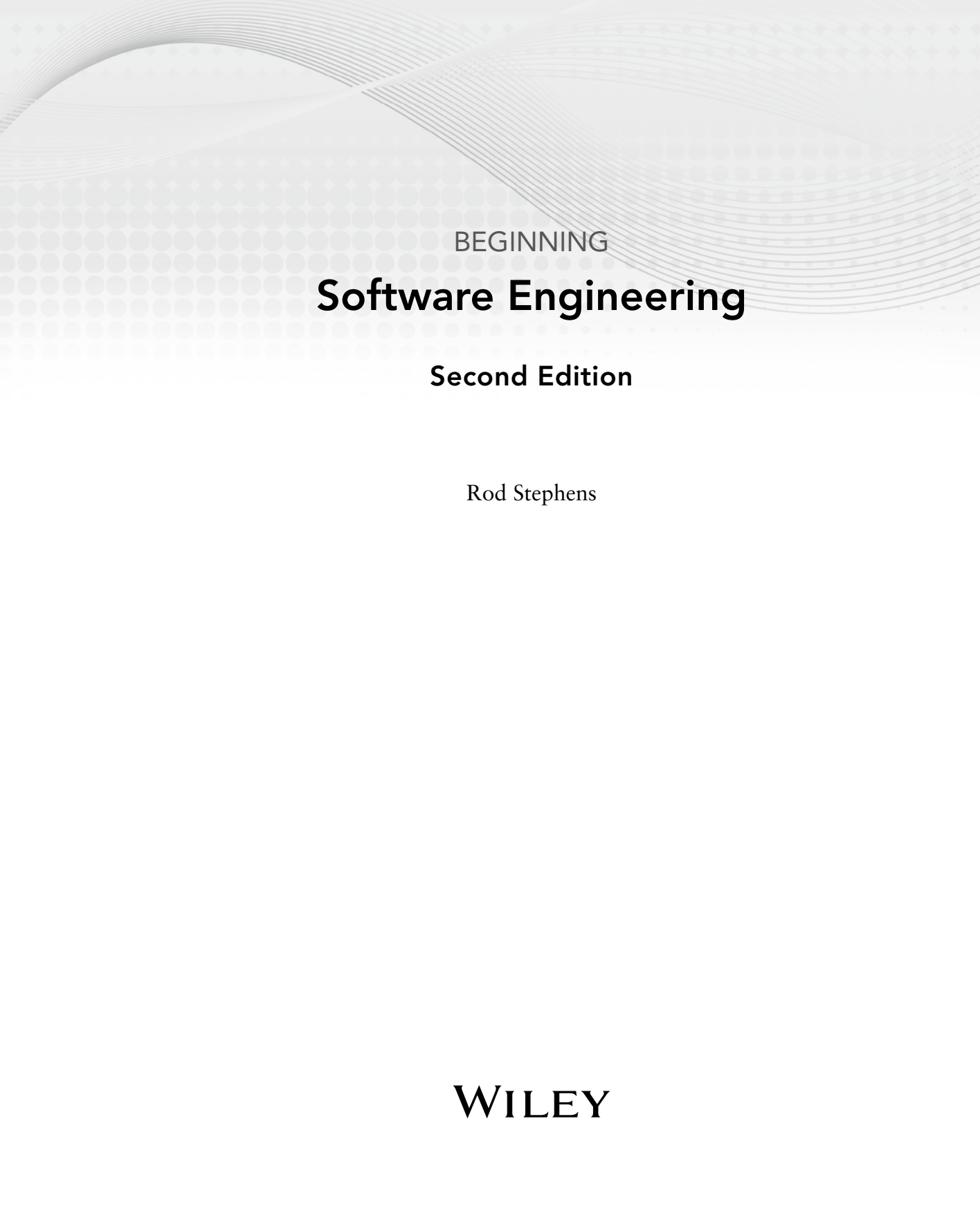
Continues

► PART III ADVANCED TOPICS

CHAPTER 20	Software Ethics	523
CHAPTER 21	Future Trends	547
APPENDIX	Solutions to Exercises	559
GLOSSARY		631
INDEX		663

BEGINNING

Software Engineering



BEGINNING

Software Engineering

Second Edition

Rod Stephens

WILEY

Copyright © 2023 by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.

Published simultaneously in Canada and the United Kingdom.

ISBN: 978-1-119-90170-9

ISBN: 978-1-119-90172-3 (ebk.)

ISBN: 978-1-119-90171-6 (ebk.)

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at www.wiley.com/go/permission.

Trademarks: Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Further, readers should be aware that websites listed in this work may have changed or disappeared between when this work was written and when it is read. Neither the publisher nor authors shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at www.wiley.com.

Library of Congress Control Number: 2022944804

Cover Image and Design: Wiley

ABOUT THE AUTHOR

Rod Stephens started out as a mathematician, but while studying at MIT, he discovered how much fun programming is and he's been programming professionally ever since. He's a long-time developer, instructor, and author who has written more than 250 magazine articles and 35 books that have been translated into many different languages.

During his career, Rod has worked on an eclectic assortment of applications in such fields as telephone switching, billing, repair dispatching, tax processing, wastewater treatment, concert ticket sales, cartography, optometry, and training for professional football teams. (That's US football, not one of the kinds with the round ball. Or the kind with three downs. Or the kind with an oval field. Or the indoor kind. Let's just say NFL and leave it at that.)

Rod's popular C# Helper website (www.csharpHelper.com) receives millions of hits per year and contains thousands of tips, tricks, and example programs for C# programmers. His VB Helper website (www.vb-helper.com) contains similar material for Visual Basic programmers.

You can contact Rod at RodStephens@csharpHelper.com.

ABOUT THE TECHNICAL EDITOR

John Mueller is a freelance author and technical editor. He has writing in his blood, having produced 122 books and more than 600 articles to date. The topics range from networking to artificial intelligence and from database management to heads-down programming. Some of his current books include discussions of data science, machine learning, and algorithms. He also writes about computer languages such as C++, C#, and Python. His technical editing skills have helped more than 70 authors refine the content of their manuscripts. John has provided technical editing services to a variety of magazines, performed various kinds of consulting, and he writes certification exams. Be sure to read John's blog at <http://blog.johnmuellerbooks.com>. You can reach John on the Internet at John@JohnMuellerBooks.com. John also has a website at www.johnmuellerbooks.com.

ACKNOWLEDGMENTS

Thanks to David Clark, Christine O'Connor, Kenyon Brown, Judy Flynn, Barath Kumar Rajasekaran, and all of the others who worked so hard to make this book possible. David was this book's project manager. You'll learn what a project manager does in Chapter 4. It's a bit different for writing a book but not as different as you might think. Many thanks for your hard work, David!

Thanks also to technical editor and longtime friend John Mueller for giving me the benefit of his valuable experience. You can see what John's up to at www.johnmuellerbooks.com.

Special thanks to Mary Brodie at <https://gearmark.blogs.com> for letting me use her quote at the beginning of Chapter 18, "Iterative Models."

CONTENTS

INTRODUCTION

xxvii

PART I: SOFTWARE ENGINEERING STEP-BY-STEP

CHAPTER 1: SOFTWARE ENGINEERING FROM 20,000 FEET	3
Requirements Gathering	4
High-Level Design	5
Low-Level Design	6
Development	6
Testing	7
Deployment	9
Maintenance	10
Wrap-Up	10
Everything All at Once	11
Summary	12
What You Learned in This Chapter	13
CHAPTER 2: BEFORE THE BEGINNING	15
Document Management	16
Historical Documents	19
Email	19
Code	22
Code Documentation	22
Application Documentation	25
Summary	26
What You Learned in This Chapter	27
CHAPTER 3: THE TEAM	29
Team Features	30
Clear Roles	30
Effective Leadership	30
Clear Goals	31
Consensus	32
Open Communication	32
Support for Risk-Taking	33

Shared Accountability	33
Informal Atmosphere	34
Trust	34
Team Roles	34
Common Roles	35
More-Specialized Roles	36
Informal Roles	36
Roles Wrap-Up	37
Team Culture	37
Interviews	40
Interview Puzzles	40
The Bottom Line	41
Physical Environment	41
Creativity	41
Office Space	43
Ergonomics	43
Work-Life Balance	45
Collaboration Software	46
Searching	46
Overload	47
Outsourcing	47
Summary	48
What You Learned in This Chapter	50
 CHAPTER 4: PROJECT MANAGEMENT	 53
Executive Support	54
Project Management	56
PERT Charts	57
Critical Path Methods	62
Gantt Charts	65
Scheduling Software	67
Predicting Times	68
Get Experience	69
Break Unknown Tasks into Simpler Pieces	70
Look for Similarities	71
Expect the Unexpected	71
Track Progress	73
Risk Management	74
Summary	76
What You Learned in This Chapter	79

CHAPTER 5: REQUIREMENTS GATHERING	81
Requirements Defined	82
Clear	82
Unambiguous	83
Consistent	84
Prioritized	84
Verifiable	88
Words to Avoid	89
Requirement Categories	89
Audience-Oriented Requirements	90
Business Requirements	90
User Requirements	90
Functional Requirements	91
Nonfunctional Requirements	92
Implementation Requirements	92
FURPS	92
FURPS+	93
Common Requirements	96
Gathering Requirements	96
Listen to Customers (and Users)	97
Use the Five Ws (and One H)	98
Who	98
What	98
When	98
Where	98
Why	99
How	99
Study Users	99
Refining Requirements	100
Copy Existing Systems	101
Clairvoyance	102
Brainstorm	103
Recording Requirements	106
UML	107
User Stories	107
Use Cases	108
Prototypes	108
Requirements Specification	109
Validation and Verification	110
Changing Requirements	110

Digital Transformation	111
What to Digitize	111
How to Digitize	112
Summary	113
What You Learned in This Chapter	116
CHAPTER 6: HIGH-LEVEL DESIGN	117
The Big Picture	118
What to Specify	119
Security	119
Hardware	120
User Interface	121
Internal Interfaces	122
External Interfaces	123
Architecture	124
Monolithic	124
Client/Server	125
Component-Based	127
Service-Oriented	128
Data-Centric	130
Event-Driven	130
Rule-Based	130
Distributed	131
Mix and Match	132
Reports	133
Other Outputs	134
Database	135
Audit Trails	136
User Access	136
Database Maintenance	137
NoSQL	137
Cloud Databases	138
Configuration Data	138
Data Flows and States	139
Training	139
UML	141
Structure Diagrams	142
Behavior Diagrams	145
Activity Diagrams	145
Use Case Diagram	146

State Machine Diagram	147
Interaction Diagrams	148
Sequence Diagram	148
Communication Diagram	150
Timing Diagram	150
Interaction Overview Diagram	151
UML Summary	151
Summary	151
What You Learned in This Chapter	152
CHAPTER 7: LOW-LEVEL DESIGN	155
Design Approaches	156
Design-to-Schedule	157
Design-to-Tools	158
Process-Oriented Design	158
Data-Oriented Design	159
Object-Oriented Design	159
Hybrid Approaches	159
High, Low, and Iterative Design	160
OO Design	160
Identifying Classes	161
Building Inheritance Hierarchies	162
Refinement	163
Generalization	165
Hierarchy Warning Signs	167
Object Composition	167
Database Design	168
Relational Databases	168
First Normal Form	170
Second Normal Form	174
Third Normal Form	176
Higher Levels of Normalization	179
When to Optimize	180
Summary	180
What You Learned in This Chapter	182
CHAPTER 8: SECURITY DESIGN	185
Security Goals	186
Security Types	186
Cybersecurity	188

Shift-Left Security	189
Malware Menagerie	189
Phishing and Spoofing	193
Social Engineering Attacks	195
Crapware	197
Password Attacks	198
User Access	201
Countermeasures	201
Cyber Insurance	202
Summary	203
What You Learned in This Chapter	207
CHAPTER 9: USER EXPERIENCE DESIGN	209
Design Mindset	210
UI vs. UX	210
UX Designers	211
Platform	212
User Skill Level	214
Beginners and Beyond	216
Configuration	217
Hidden Configuration	218
Models	219
Metaphors and Idioms	220
Case Study: Microsoft Word	221
Design Guidelines	225
Allow Exploration	225
Make the Interface Immutable	227
Support Commensurate Difficulty	227
Avoid State	228
Make Similar Things Similar	228
Provide Redundant Commands	230
Do the Right Thing	231
Show Qualitative Data, Explain Quantitative Data	232
Give Forms Purpose	232
Gather All Information at Once	233
Provide Reasonable Performance	234
Only Allow What's Right	235
Flag Mistakes	235
Form Design	236
Use Standard Controls	236
Decorating	237

Displaying	237
Arranging	237
Commanding	238
Selecting	238
Entering	239
Display Five Things	240
Arrange Controls Nicely	241
Summary	241
What You Learned in This Chapter	242
CHAPTER 10: PROGRAMMING	245
Tools	246
Hardware	246
Network	247
Development Environment	248
Source Code Control	249
Profilers	249
Static Analysis Tools	249
Testing Tools	249
Source Code Formatters	250
Refactoring Tools	250
Training	250
Collaboration Tools	250
Algorithms	251
Top-Down Design	252
Programming Tips and Tricks	255
Be Alert	255
Write for People, Not the Computer	255
Comment First	256
Write Self-Documenting Code	259
Keep It Small	259
Stay Focused	261
Avoid Side Effects	261
Validate Results	262
Practice Offensive Programming	264
Use Exceptions	266
Write Exception Handlers First	266
Don't Repeat Code	267
Defer Optimization	267
Summary	269
What You Learned in This Chapter	270

CHAPTER 11: ALGORITHMS	273
Algorithm Study	274
Algorithmic Approaches	275
Decision Trees	275
Knapsack	275
The Eight Queens Problem	276
Exhaustive Search	277
Backtracking	278
Pruning Trees	279
Branch and Bound	279
Heuristics	280
Greedy	281
Divide and Conquer	282
Recursion	283
Dynamic Programming	285
Caching	287
Randomization	287
Monte Carlo Algorithms	287
Las Vegas Algorithms	288
Atlantic City Algorithms	289
State Diagrams	289
Design Patterns	290
Creational Patterns	291
Structural Patterns	291
Behavioral Patterns	292
Design Pattern Summary	293
Parallel Programming	293
Artificial Intelligence	295
Definitions	295
Learning Systems	296
Natural Language Processing	297
Artificial Neural Network	297
Deep Learning	297
Expert System	298
Artificial General Intelligence	298
Algorithm Characteristics	301
Summary	302
What You Learned in This Chapter	304

CHAPTER 12: PROGRAMMING LANGUAGES	307
The Myth of Picking a Language	308
Language Generations	311
First Generation	311
Second Generation	311
Third Generation (3GL)	312
Fourth Generation	313
Fifth Generation	314
Sixth Generation	314
IDEs	315
Language Families	316
Assembly	316
Imperative	317
Procedural	317
Declarative	318
Object-Oriented	318
Functional	319
Specialized	319
Language Family Summary	319
The Best Language	319
Summary	323
What You Learned in This Chapter	324
CHAPTER 13: TESTING	327
Testing Goals	329
Reasons Bugs Never Die	330
Diminishing Returns	330
Deadlines	330
Consequences	330
It's Too Soon	330
Usefulness	331
Obsolescence	331
It's Not a Bug	331
It Never Ends	332
It's Better Than Nothing	333
Fixing Bugs Is Dangerous	333
Which Bugs to Fix	334
Levels of Testing	334
Unit Testing	335

Integration Testing	336
Regression Testing	337
Automated Testing	337
Component Interface Testing	338
System Testing	339
Acceptance Testing	340
Other Testing Categories	341
Testing Techniques	342
Exhaustive Testing	342
Black-Box Testing	343
White-Box Testing	344
Gray-Box Testing	344
Testing Habits	345
Test and Debug When Alert	345
Test Your Own Code	346
Have Someone Else Test Your Code	346
Fix Your Own Bugs	348
Think Before You Change	349
Don't Believe in Magic	349
See What Changed	350
Fix Bugs, Not Symptoms	350
Test Your Tests	350
How to Fix a Bug	351
Estimating Number of Bugs	351
Tracking Bugs Found	352
Seeding	353
The Lincoln Index	353
Summary	355
What You Learned in This Chapter	357
 CHAPTER 14: DEPLOYMENT	 359
Scope	360
The Plan	361
Cutover	362
Staged Deployment	362
Gradual Cutover	363
Incremental Deployment	365
Parallel Testing	365
Deployment Tasks	365

Deployment Mistakes	366
Summary	368
What You Learned in This Chapter	370
CHAPTER 15: METRICS	371
Wrap Party	372
Defect Analysis	372
Species of Bugs	373
Discoverer	373
Severity	374
Creation Time	374
Age at Fix	374
Task Type	375
Defect Database	376
Ishikawa Diagrams	376
Software Metrics	379
Qualities of Good Attributes and Metrics	381
Using Metrics	382
Process Metrics	384
Project Metrics	384
Things to Measure	385
Size Normalization	387
Function Point Normalization	389
Count Function Point Metrics	390
Multiply by Complexity Factors	391
Calculate Complexity Adjustment Value	392
Calculate Adjusted FP	394
Summary	395
What You Learned in This Chapter	398
CHAPTER 16: MAINTENANCE	401
Maintenance Costs	402
Task Categories	404
Perfective Tasks	404
Feature Improvements	406
New Features	406
The Second System Effect	407
Adaptive Tasks	408
Corrective Tasks	410

Preventive Tasks	414
Clarification	414
Code Reuse	415
Improved Flexibility	416
Bug Swarms	417
Bad Programming Practices	417
Individual Bugs	418
Not Invented Here	418
Task Execution	419
Summary	420
What You Learned in This Chapter	423

PART II: PROCESS MODELS

CHAPTER 17: PREDICTIVE MODELS 427

Model Approaches	428
Prerequisites	428
Predictive and Adaptive	429
Success and Failure Indicators for Predictive Models	430
Advantages and Disadvantages of Predictive Models	431
Waterfall	432
Waterfall with Feedback	433
Sashimi	434
Incremental Waterfall	436
V-model	438
Software Development Life Cycle	439
Summary	442
What You Learned in This Chapter	444

CHAPTER 18: ITERATIVE MODELS 445

Iterative vs. Predictive	446
Iterative vs. Incremental	448
Prototypes	449
Types of Prototypes	451
Pros and Cons	451
Spiral	453
Clarifications	455
Pros and Cons	456
Unified Process	457

Pros and Cons	459
Rational Unified Process	459
Cleanroom	460
Cowboy Coding	461
Summary	461
What You Learned in This Chapter	463
CHAPTER 19: RAD	465
RAD Principles	467
James Martin RAD	470
Agile	471
Self-Organizing Teams	473
Agile Techniques	474
Communication	474
Incremental Development	475
Focus on Quality	478
XP	478
XP Roles	479
XP Values	480
XP Practices	481
Have a Customer On-Site	481
Play the Planning Game	482
Use Stand-Up Meetings	483
Make Frequent Small Releases	483
Use Intuitive Metaphors	484
Keep Designs Simple	484
Defer Optimization	484
Refactor When Necessary	485
Give Everyone Ownership of the Code	485
Use Coding Standards	486
Promote Generalization	486
Use Pair Programming	486
Test Constantly	486
Integrate Continuously	486
Work Sustainably	487
Use Test-Driven and Test-First Development	487
Scrum	488
Scrum Roles	489
Scrum Sprints	490
Planning Poker	491

Burndown	492
Velocity	494
Lean	494
Lean Principles	494
Crystal	495
Crystal Clear	498
Crystal Yellow	498
Crystal Orange	499
Feature-Driven Development	500
FDD Roles	501
FDD Phases	502
Develop a Model	502
Build a Feature List	502
Plan by Feature	503
Design by Feature	503
Build by Feature	504
FDD Iteration Milestones	504
Disciplined Agile Delivery	506
DAD Principles	506
DAD Roles	506
DAD Phases	507
Dynamic Systems Development Method	508
DSDM Phases	508
DSDM Principles	510
DSDM Roles	511
Kanban	512
Kanban Principles	513
Kanban Practices	513
Kanban Board	514
Summary	515
What You Learned in This Chapter	517

PART III: ADVANCED TOPICS

CHAPTER 20: SOFTWARE ETHICS	523
Ethical Behavior	524
IEEE-CS/ACM	524
ACS	525
CPSR	526
Business Ethics	527
NADA	528

Hacker Ethics	529
Hacker Terms	530
Responsibility	531
Gray Areas	533
Software Engineering Dilemmas	535
Misusing Data and the Temptation of Free Data	535
Disruptive Technology	536
Algorithmic Bias	537
False Confidence	537
Lack of Oversight	538
Getting Paid	539
Thought Experiments	539
The Tunnel Problem	540
The Trolley Problem	542
Summary	544
What You Learned in This Chapter	545
 CHAPTER 21: FUTURE TRENDS	 547
 Security	 548
UX/UI	549
Code Packaging	550
Cloud Technology	551
Software Development	552
Algorithms	553
Tech Toys	554
Summary	555
What You Learned in This Chapter	556
 APPENDIX: SOLUTIONS TO EXERCISES	 559
 GLOSSARY	 631
INDEX	663

INTRODUCTION

Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the universe trying to build bigger and better idiots. So far the universe is winning.

—Rick Cook

With modern development tools, it's easy to sit down at the keyboard and bang out a working program with no previous design or planning, and that's fine under some circumstances. My VB Helper (www.vb-helper.com) and C# Helper (www.csharpHelper.com) websites contain thousands of example programs written in Visual Basic and C#, respectively, and built using exactly that approach. I had an idea (or someone asked me a question) and I pounded out a quick example.

Those types of programs are fine if you're the only one using them and then for only a short while. They're also okay if, as on my websites, they're intended only to demonstrate a programming technique and they never leave the confines of the programming laboratory.

If this kind of slap-dash program escapes into the wild, however, the result can be disastrous. At best, nonprogrammers who use these programs quickly become confused. At worst, they can wreak havoc on their computers and even on those of their friends and coworkers.

Even experienced developers sometimes run afoul of these half-baked programs. I know someone (I won't give names, but I also won't say it wasn't me) who wrote a simple recursive script to delete the files in a directory hierarchy. Unfortunately, the script recursively climbed its way to the top of the directory tree and then started cheerfully deleting every file in the system. The script ran for only about five seconds before it was stopped, but it had already trashed enough files that the operating system had to be reinstalled from scratch. (Actually, some developers believe reinstalling the operating system every year or so is character-building. If you agree, then perhaps this approach isn't so bad.)

I know another experienced developer who, while experimenting with Windows system settings, managed to set every system color to black. The result was a black cursor over a black desktop, displaying black windows with black borders, menus, and text. This person (who wasn't me this time) eventually managed to fix things by rebooting and using another computer that wasn't color-impaired to walk through the process of fixing the settings using only keyboard accelerators. It was a triumph of cleverness, but I suspect she would have rather skipped the whole episode and had her two wasted days back.

For programs that are more than a few dozen lines long, or that will be given to unsuspecting end users, this kind of free-spirited development approach simply won't do. To produce applications that are effective, safe, and reliable, you can't just sit down and start typing. You need a plan. You need . . . <drumroll> . . . software engineering.

This book describes software engineering. It explains what software engineering is and how it helps produce applications that are effective, flexible, and robust enough for use in real-world situations.

This book won't make you an expert systems analyst, software architect, project manager, or programmer, but it explains what those people do and why they are necessary for producing high-quality software. It also gives you the tools that you need to start. You won't rush out and lead a 1,000-person effort to build a new air traffic control system for the FAA, but it can help you work effectively in small-scale and large-scale development projects. (It can also help you understand what a prospective employer means when he says, "Yeah, we mostly use scrum with a few extra XP techniques thrown in.")

WHAT IS SOFTWARE ENGINEERING?

A formal definition of software engineering might sound something like, "An organized, analytical approach to the design, development, use, and maintenance of software."

More intuitively, software engineering is everything that you need to do to produce successful software. It includes the steps that take a raw, possibly nebulous idea and turn it into a powerful and intuitive application that can be enhanced to meet changing customer needs for years to come.

You might be tempted to restrict software engineering to mean only the beginning of the process, when you perform the application's design. After all, an aerospace engineer designs planes but doesn't build them or tack on a second passenger cabin if the first one becomes full. (Although I guess a space shuttle riding piggyback on a 747 sort of achieved that goal.)

One of the big differences between software engineering and aerospace engineering (or most other kinds of engineering) is that software isn't physical. It exists only in the virtual world of the computer. That means it's easy to make changes to any part of a program even after it is completely written. In contrast, if you wait until a bridge is finished and then tell your structural engineer that you've decided to add two extra lanes and lift it three feet higher above the water, there's a good chance he'll cackle wildly and offer you all sorts of creative but impractical suggestions for exactly what you can do with your two extra lanes.

The flexibility granted to software by its virtual nature is both a blessing and a curse. It's a blessing because it lets you refine the program during development to better meet user needs, add new features to take advantage of opportunities discovered during implementation, and make modifications to meet evolving business requirements. Some applications even allow users to write scripts to perform new tasks never envisioned by the developers. That type of flexibility isn't possible in other types of engineering.

Unfortunately, the flexibility that allows you to make changes throughout a software project's life cycle also lets you mess things up at any point during development. Adding a new feature can break existing code or turn a simple, elegant design into a confusing mess. Constantly adding, removing, and modifying features during development can make it impossible for different parts of the system to work together. In some cases, it can even make it impossible to tell when the project is finished.