

Making Software

What Really Works, and Why We Believe It

Edited by Andy Oram and Greg Wilson

O'REILLY[®]

Beijing • Cambridge • Farnham • Koln • Sebastopol • Tokyo

CONTENTS

PREFACE	xi
<u>Part One</u> GENERAL PRINCIPLES OF SEARCHING FOR AND USING EVIDENCE	
1 THE QUEST FOR CONVINCING EVIDENCE	3
<i>by Tim Menzies and Forrest Shull</i>	
In the Beginning	H
The Slate of Evidence Today	H
Change We Can Believe In	8
The Effect of Context	10
Looking Toward the Future	11
2 CREDIBILITY, OR WHY SHOULD I INSIST ON BEING CONVINCED?	17
<i>by Lutz Prechelt and Marian Petre</i>	
How Evidence Turns Up in Software Engineering	17
Credibility and Relevance	19
Aggregating Evidence	22
Types of Evidence and Their Strengths and Weaknesses	25
Society, Culture, Software Engineering, and You	32
Acknowledgments	33
3 WHAT WE CAN LEARN FROM SYSTEMATIC REVIEWS	35
<i>by Barbara Kitchenham</i>	
An Overview of Systematic Reviews	36
The Strengths and Weaknesses of Systematic Reviews	39
Systematic Reviews in Software Engineering	HH
Conclusion	49
1 UNDERSTANDING SOFTWARE ENGINEERING THROUGH QUALITATIVE METHODS	55
<i>by Andrew Ko</i>	
What Are Qualitative Methods?	56
Reading Qualitative Research	58
Using Qualitative Methods in Practice	60
Generalizing from Qualitative Results	62
Qualitative Methods Are Systematic	62
5 LEARNING THROUGH APPLICATION: THE MATURING OF THE QIP IN THE SEL	65
<i>by Victor R. Basili</i>	
What Makes Software Engineering Uniquely Hard to Research	65

	A Realistic Approach to Empirical Research	66
	The NASA Software Engineering Laboratory: A Vibrant Testbed for Empirical Research	67
	The Quality Improvement Paradigm	69
	Conclusion	75
6	PERSONALITY, INTELLIGENCE, AND EXPERTISE: IMPACTS ON SOFTWARE DEVELOPMENT <i>by Jo E. Hannay</i>	79
	How to Recognize Good Programmers	81
	Individual or Environment	95
	Concluding Remarks	101
7	WHY IS IT SO HARD TO LEARN TO PROGRAM? <i>by Mark Guzdial</i>	III
	Do Students Have Difficulty Learning to Program?	112
	What Do People Understand Naturally About Programming?	III
	Making the Tools Better by Shifting to Visual Programming	117
	Contextualizing for Motivation	118
	Conclusion: A Fledgling Field	121
8	BEYOND LINES OF CODE: DO WE NEED MORE COMPLEXITY METRICS? <i>by Israel Herraiz and Ahmed E. Hassan</i>	125
	Surveying Software	126
	Measuring the Source Code	127
	A Sample Measurement	128
	Statistical Analysis	133
	Some Comments on the Statistical Methodology	139
	So Do We Need More Complexity Metrics?	140
Part Two SPECIFIC TOPICS IN SOFTWARE ENGINEERING		
9	AN AUTOMATED FAULT PREDICTION SYSTEM <i>by Elaine J. Weyuker and Thomas J. Ostrand</i>	145
	Fault Distribution	146
	Characteristics of Faulty Files	149
	Overview of the Prediction Model	150
	Replication and Variations of the Prediction Model	151
	Building a Tool	157
	The Warning Label	157
10	ARCHITECTING: HOW MUCH AND WHEN? <i>by Barry Boehm</i>	161
	Does the Cost of Fixing Software Increase over the Project Life Cycle?	162
	How Much Architecting Is Enough?	162
	Using What We Can Learn from Cost-to-Fix Data About the Value of Architecting	165
	So How Much Architecting Is Enough?	178
	Does the Architecting Need to Be Done Up Front?	181
	Conclusions	182

11	CONWAY'S COROLLARY <i>by Christian Bird</i>	187
	Conway's Law	187
	Coordination, Congruence, and Productivity	189
	Organizational Complexity Within Microsoft	194
	Chapels in the Bazaar of Open Source Software	201
	Conclusions	205
12	HOW EFFECTIVE IS TEST-DRIVEN DEVELOPMENT? <i>by Burak Turhan, Lucas Layman, Madeline D'Ep, Hakan Erdogmus, and Forrest Shull</i>	207
	The TDD Pill—What Is It?	208
	Summary of Clinical TDD Trials	209
	The Effectiveness of TDD	211
	Enforcing Correct TDD Dosage in Trials	214
	Cautions and Side Effects	215
	Conclusions	216
	Acknowledgments	217
13	WHY AREN'T MORE WOMEN IN COMPUTER SCIENCE? <i>by Michele A. Whitecraft and Wendy M. Williams</i>	221
	Why So Few Women?	222
	Should We Care?	227
	Conclusion	234
11	TWO COMPARISONS OF PROGRAMMING LANGUAGES <i>by Lutz Prechelt</i>	239
	A Language Shoot-Out over a Peculiar Search Algorithm	240
	Platform Forms: Web Development Technologies and Cultures	248
	So What?	257
15	QUALITY WARS: OPEN SOURCE VERSUS PROPRIETARY SOFTWARE <i>by Diomidis Spinellis</i>	259
	Past Skirmishes	260
	The Battlefield	261
	Into the Battle	265
	Outcome and Aftermath	286
	Acknowledgments and Disclosure of Interest	289
16	CODE TALKERS <i>by Robert DeLine</i>	295
	A Day in the Life of a Programmer	295
	What Is All This Talk About?	298
	A Model for Thinking About Communication	307
17	PAIR PROGRAMMING <i>by Laurie Williams</i>	311
	A History of Pair Programming	312

Pair Programming in an Industrial Setting	314
Pair Programming in an Educational Setting	317
Distributed Pair Programming	319
Challenges	320
Lessons Learned	321
Acknowledgments	322
 MODERN CODE REVIEW	 329
<i>by Jason Cohen</i>	
Common Sense	329
A Developer Does a Little Code Review	330
Group Dynamics	334
Conclusion	336
 A COMMUNAL WORKSHOP OR DOORS THAT CLOSE?	 339
<i>by Jorge Aranda</i>	
Doors That Close	339
A Communal Workshop	342
Work Patterns	345
One More Thing...	347
 IDENTIFYING AND MANAGING DEPENDENCIES IN GLOBAL SOFTWARE DEVELOPMENT	 349
<i>by Marcelo Cataldo</i>	
Why Is Coordination a Challenge in GSD?	350
Dependencies and Their Socio-Technical Duality	351
From Research to Practice	362
Future Directions	366
 HOW EFFECTIVE IS MODULARIZATION?	 373
<i>by Neil Thomas and Gail Murphy</i>	
The Systems	374
What Is a Change?	376
What Is a Module?	381
The Results	383
Threats to Validity	389
Summary	390
 THE EVIDENCE FOR DESIGN PATTERNS	 393
<i>by Waller Tichy</i>	
Design Pattern Examples	394
Why Might Design Patterns Work?	397
The First Experiment: Testing Pattern Documentation	398
The Second Experiment: Comparing Pattern Solutions to Simpler Ones	403
The Third Experiment: Patterns in Team Communication	407
Lessons Learned	410
Conclusions	412
Acknowledgments	413

EVIDENCE-BASED FAILURE PREDICTION <i>by Nachiappan Nagappan and Thomas Ball</i>	415
Introduction	416
Code Coverage	417
Code Churn	418
Code Complexity	421
Code Dependencies	422
People and Organizational Measures	423
Integrated Approach for Prediction of Failures	426
Summary	430
Acknowledgments	432
THE ART OF COLLECTING BUG REPORTS <i>by Rahul Premraj and Thomas Zimmermann</i>	435
Good and Bad Bug Reports	436
What Makes a Good Bug Report?	437
Survey Results	439
Evidence for an Information Mismatch	441
Problems with Bug Reports	444
The Value of Duplicate Bug Reports	445
Not All Bug Reports Get Fixed	448
Conclusions	449
Acknowledgments	450
WHERE DO MOST SOFTWARE FLAWS COME FROM? <i>by Dewayne Perry</i>	453
Studying Software Flaws	454
Context of the Study	455
Phase 1: Overall Survey	456
Phase 2: Design/Code Fault Survey	462
What Should You Believe About These Results?	486
What Have We Learned?	490
Acknowledgments	492
NOVICE PROFESSIONALS: RECENT GRADUATES IN A FIRST SOFTWARE ENGINEERING JOB <i>by Andrew Begel and Beth Simon</i>	495
Study Methodology	497
Software Development Task	501
Strengths and Weaknesses of Novice Software Developers	505
Reflections	507
Misconceptions That Hinder Learning	509
Reflecting on Pedagogy	510
Implications for Change	512
MINING YOUR OWN EVIDENCE <i>by Kim Sebastian Herzij and Andreas Zeller</i>	517
What Is There to Mine?	518

	Designing a Study	518
	A Mining Primer	519
	Where to Go from Here	526
	Acknowledgments	528
28	COPY-PASTE AS A PRINCIPLED ENGINEERING TOOL <i>by Michael Godfrey and Cory Kapser</i>	531
	An Example of Code Cloning	532
	Detecting Clones in Software	533
	Investigating the Practice of Code Cloning	535
	Our Study	540
	Conclusions	543
29	HOW USABLE ARE YOUR APIS? <i>by Steven Clarke</i>	545
	Why Is It Important to Study API Usability?	546
	First Attempts at Studying API Usability	548
	If At First You Don't Succeed...	552
	Adapting to Different Work Styles	559
	Conclusion	563
30	WHAT DOES 10X MEAN? MEASURING VARIATIONS IN PROGRAMMER PRODUCTIVITY <i>by Steve McConnell</i>	567
	Individual Productivity Variation in Software Development	567
	Issues in Measuring Productivity of Individual Programmers	569
	Team Productivity Variation in Software Development	571
A	CONTRIBUTORS	575
	INDEX	587