

Extracted from:

# The ThoughtWorks Anthology

## Essays on Software Technology and Innovation

---

This PDF file contains pages extracted from The ThoughtWorks Anthology, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

**Note:** This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2008 The Pragmatic Programmers, LLC.

All rights reserved.

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, or otherwise, without the prior consent of the publisher.

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>11</b>
<b>2</b>	<b>Solving the Business Software “Last Mile”</b>	<b>15</b>
	<i>by Roy Singham and Michael Robinson</i>	
2.1	The Source of the “Last Mile” Problem . . . . .	15
2.2	Understanding the Problem . . . . .	16
2.3	Solving the “Last Mile” Problem . . . . .	18
2.4	People . . . . .	18
2.5	Automation . . . . .	19
2.6	Design for Automated Testing of Nonfunctional Requirements . . . . .	20
2.7	Decouple Design from Production Environment . . . . .	22
2.8	Versionless Software . . . . .	23
<b>3</b>	<b>One Lair and Twenty Ruby DSLs</b>	<b>25</b>
	<i>by Martin Fowler</i>	
3.1	My Lair Example . . . . .	25
3.2	Using Global Functions . . . . .	28
3.3	Using Objects . . . . .	31
3.4	Using Closures . . . . .	37
3.5	Evaluation Context . . . . .	38
3.6	Literal Collections . . . . .	41
3.7	Dynamic Reception . . . . .	46
3.8	Final Thoughts . . . . .	48
<b>4</b>	<b>The Lush Landscape of Languages</b>	<b>49</b>
	<i>by Rebecca J. Parsons</i>	
4.1	Introduction . . . . .	49
4.2	The Specimens . . . . .	49
4.3	The Variety of Varieties . . . . .	53
4.4	The Tree of Life for Languages . . . . .	57
4.5	That’s All Very Interesting, But Why Should You Care? . . . . .	59

<b>5 Polyglot Programming</b>	<b>60</b>
<i>by Neal Ford</i>	
5.1 Polyglot Programming . . . . .	61
5.2 Reading Files the Groovy Way . . . . .	61
5.3 JRuby and isBlank . . . . .	63
5.4 Jaskell and Functional Programming . . . . .	64
5.5 Testing Java . . . . .	67
5.6 Polyglot Programming the Future . . . . .	69
<b>6 Object Calisthenics</b>	<b>70</b>
<i>by Jeff Bay</i>	
6.1 Nine Steps to Better Software Design Today . . . . .	70
6.2 The Exercise . . . . .	71
6.3 Conclusion . . . . .	79
<b>7 What Is an Iteration Manager Anyway?</b>	<b>81</b>
<i>by Tiffany Lentz</i>	
7.1 What Is an Iteration Manager? . . . . .	81
7.2 What Makes a Good Iteration Manager? . . . . .	82
7.3 What an Iteration Manager Is Not . . . . .	83
7.4 The Iteration Manager and the Team . . . . .	84
7.5 The Iteration Manager and the Customer . . . . .	85
7.6 The Iteration Manager and the Iteration . . . . .	86
7.7 The Iteration Manager and the Project . . . . .	87
7.8 Conclusion . . . . .	88
<b>8 Project Vital Signs</b>	<b>89</b>
<i>by Stelios Pantazopoulos</i>	
8.1 Project Vital Signs . . . . .	89
8.2 Project Vital Signs vs. Project Health . . . . .	90
8.3 Project Vital Signs vs. Information Radiator . . . . .	90
8.4 Project Vital Sign: Scope Burn-Up . . . . .	91
8.5 Project Vital Sign: Delivery Quality . . . . .	94
8.6 Project Vital Sign: Budget Burn-Down . . . . .	95
8.7 Project Vital Sign: Current State of Implementation . . . . .	97
8.8 Project Vital Sign: Team Perceptions . . . . .	100
<b>9 Consumer-Driven Contracts: A Service Evolution Pattern</b>	<b>101</b>
<i>by Ian Robinson</i>	
9.1 Evolving a Service: An Example . . . . .	103
9.2 Schema Versioning . . . . .	104

9.3	Breaking Changes . . . . .	109
9.4	Consumer-Driven Contracts . . . . .	111
<b>10</b>	<b>Domain Annotations</b>	<b>121</b>
	<i>by Erik Doernenburg</i>	
10.1	Domain-Driven Design Meets Annotations . . . . .	121
10.2	Case Study: Leroy's Lorries . . . . .	126
10.3	Summary . . . . .	140
<b>11</b>	<b>Refactoring Ant Build Files</b>	<b>142</b>
	<i>by Julian Simpson</i>	
11.1	Introduction . . . . .	142
11.2	Ant Refactoring Catalog . . . . .	144
11.3	Summary . . . . .	171
11.4	References . . . . .	171
11.5	Resources . . . . .	171
<b>12</b>	<b>Single-Click Software Release</b>	<b>172</b>
	<i>by Dave Farley</i>	
12.1	Continuous Build . . . . .	172
12.2	Beyond Continuous Build . . . . .	173
12.3	Full Lifecycle Continuous Integration . . . . .	174
12.4	The Check-in Gate . . . . .	175
12.5	The Acceptance Test Gate . . . . .	177
12.6	Preparing to Deploy . . . . .	177
12.7	Subsequent Test Stages . . . . .	180
12.8	Automating the Process . . . . .	181
12.9	Conclusion . . . . .	181
<b>13</b>	<b>Agile vs. Waterfall Testing for Enterprise Web Apps</b>	<b>183</b>
	<i>by Kristan Vingrys</i>	
13.1	Introduction . . . . .	183
13.2	Testing Life Cycle . . . . .	184

13.3	Types of Testing . . . . .	187
13.4	Environments . . . . .	193
13.5	Issue Management . . . . .	196
13.6	Tools . . . . .	197
13.7	Reports and Metrics . . . . .	198
13.8	Testing Roles . . . . .	199
13.9	References . . . . .	201
<b>14</b>	<b>Pragmatic Performance Testing</b>	<b>202</b>
	<i>by James Bull</i>	
14.1	What Is Performance Testing? . . . . .	202
14.2	Requirements Gathering . . . . .	203
14.3	Running the Tests . . . . .	208
14.4	Communication . . . . .	214
14.5	Process . . . . .	216
14.6	Summary . . . . .	218
	<b>Bibliography</b>	<b>219</b>
	<b>Index</b>	<b>220</b>

## Chapter 1

# Introduction

---

ThoughtWorks is a collection of passionate, driven, intelligent individuals that delivers custom applications and no-nonsense consulting. Ask a ThoughtWorker what they like most about the company, and they will likely say it is the other ThoughtWorkers they get to meet, work with, and learn from. We're a mixture of geeks, managers, analysts, programmers, testers, and operations folks with varied cultural, ethnic, and educational backgrounds. This diversity of background and perspective, coupled with a passion for ideas that we share, can result in some pretty lively debates.

We have created a successful company with nearly 1,000 smart, opinionated people in six countries organized with little hierarchy and a fanatical commitment to transparency. Of course, our definition of success is not the typical one either; success must encompass client satisfaction, impact on our industry, and impact on our society. We do aim high.

The voices of many ThoughtWorkers are heard in the blogosphere, on the conference circuit, on the Web, and on the bookshelves. Indeed, part of our commitment to excellence involves ruthlessly critiquing what we've done and how we've done it to see how to improve it the next time. We're a tough bunch to satisfy. Once we've learned something, we want to tell others about it.

Our battle scars come from myriad projects in different domains, technologies, and platform choices. Although we do think (a lot) about what we do, that thinking is grounded in the real world of delivering lots of software for people. There's purity to our function that has allowed us to focus on developing software.

One doesn't generally pay a consultant to sit in meetings discussing the new HR policies, so our workdays are far more focused on delivering software than most IT professionals, resulting in a combination of pragmatism and rigor.

This anthology provides a great snapshot into the incredibly diverse set of IT problems on which ThoughtWorkers are working. This anthology strives to do more than simply present a few ways to produce better software; it grapples with the problems of realizing actual business value from the IT efforts that organizations take on. Roy's opening essay sets the tone with his call to arms for bringing about a change in the "last mile" of getting a system into the production environment. His program is broad and ambitious—nothing less than making those operational and deployment issues as core to the development process as the requirements gathering and coding itself. By remembering that success is not merely getting your code to pass your QA department and have it ready to toss over the wall at an operations team that deals with production, deployment, and the like, the team building the software knows they're not "done" until they've seen the software to the end. And Roy's advocacy goes past simply some clever redefinitions of *completion* and *success*. He calls for a rethinking of how and when stakeholders get involved. All the genius that has gone into making tools better for the coding process (for example, tools for automated builds and scripted testing, as well as refactoring) can be applied to tackling much of the "last mile" problem.

As you read through the collection, you'll see that his call gets answered repeatedly. For example, James takes on performance testing, an area that is habitually neglected and put off until the late stages of a project, when so many design decisions have been baked into the code that undoing them without damaging the hard-won working business functionality for the sake of tackling performance feels like an undertaking in violation of the Second Law of Thermodynamics. James takes a suitably pragmatic approach, not simply arguing that we need the performance requirements up front (who can argue with this?) but discussing ways to get useful requirements from the stakeholders. He doesn't simply say "test early!" but actually discusses how and where these tests can be run.

Julian takes on Ant refactoring by cataloging a large number of standard refactorings and then providing clear examples for each. His essay is an excellent reference for anyone dealing with a growing and evol-

ing build script. Dave's essay provides nice bookend symmetry to Roy's opening with his outlining of the conceptual framework around single-click deployment. He takes on some big issues, such as managing the large, unwieldy binaries that get generated and integration in the heterogeneous environments in which software is typically deployed. All the techniques that work to make business-software development effective will eventually migrate into the world of the deployment tools. Dave's essay takes that program forward.

Stelios's essay takes on communication techniques for conveying project health. He puts forth some metrics, both objective and subjective, and discusses effective ways to present them so that everyone involved has the same "dashboard" to work from every day. He's bringing the visibility of the project's vital signs to as many stakeholders as possible. This connects to another notion: a sort of project anthropology. Tiffany's essay reads like Margaret Mead reporting on her findings in Samoa. She has stumbled upon a whole new kind of project team member, the iteration manager, and tells us about how it fits into the tribe. She sees a chance to address how to organize the team a little differently to make it more effective, and hence we have a role to help work through this. Jeff's "nine rules of thumb" essay reminds me of some master talking to disciples about the Tao of programming. The rules are simple and elegant and maddeningly hard to adhere to (especially because they require any coder to "unlearn" so many habits). Rebecca's essay feels to me like she sneaks in a strong stance on the issue of "language wars" by starting out as an engaging read on classifying various languages. At first you read along, imagining Linnaeus strolling through a garden, looking at particular characteristics of the plants he sees, and then generalizing them to a framework for classifying any plant he comes along in the future. Rebecca lays down a great foundation. But her surprise comes at the end: this isn't just some survey course in some languages currently in vogue but instead a demonstration of the diversity of tools out there and that any particular "Java vs. .NET" language dispute is just the latest iteration in a never-ending conversation. But what matters is knowing what kind of problem you're trying to solve and what kind of tools you have at your disposal for tackling them. I feel like she came into a cluttered workshop, sorted the wrenches from the hammers, and put them in drawers with some labels that tell you what the items within are good for.

The remaining essays are a lot more technically focused but demonstrate more of the diversity of talent that I get to call fellow co-workers.



Ian lays out a comprehensive approach for thinking about SOA contracts that are consumer, rather than customer, driven. His essay takes another whack at the eternal problem of how to build and evolve shared services that need to adapt over time to changing business needs and do so in a way that doesn't cripple the existing consumers of that service. And Erik considers a similar problem. In a well-designed system, you decouple the domain model from infrastructure layers, but this requires that the infrastructure layer use the metadata present in the domain model. Some implicit metadata can be gleaned from what's in there such as the choice of classes to represent certain domain elements, but it doesn't really provide enough information for really rich stuff like validations. Some modern languages such as Java and C# provide for more metadata in the form annotations and attributes, and Erik explores how to exploit these features through a case study. And Martin's playful romp through various DSLs for evil megalomaniacs reminded me of when I was learning C so long ago. My reference was Kernighan and Ritchie, and I watched in amazement as they worked through a few iterations of a string copy function to bring it to a level of simplicity and elegance that seemed to elude me through all my subsequent programming efforts.

The threads of connection are everywhere in this anthology. These essays explore an ecosystem of IT problems and yet link together in all sorts of obvious and surprising ways. The breadth of topics and variety of approaches for solving them reflect the health of an environment of ideas that exists at the organization that all these authors are part of. Seeing a slice of it in the form of this collection leaves me hungry to see what else we're capable of doing.

*Mike Aguilar (Vice President, ThoughtWorks)*

*February 15, 2008*

# The Pragmatic Bookshelf

---

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

## Visit Us Online

---

### ThoughtWorks Anthology's Home Page

<http://pragprog.com/titles/twa>

Source code from this book, errata, and other resources. Come give us feedback, too!

### Register for Updates

<http://pragprog.com/updates>

Be notified when updates and new books become available.

### Join the Community

<http://pragprog.com/community>

Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

### New and Noteworthy

<http://pragprog.com/news>

Check out the latest pragmatic developments in the news.

## Buy the Book

---

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: [pragprog.com/titles/twa](http://pragprog.com/titles/twa).

## Contact Us

---

Phone Orders:	1-800-699-PROG (+1 919 847 3884)
Online Orders:	<a href="http://www.pragprog.com/catalog">www.pragprog.com/catalog</a>
Customer Service:	<a href="mailto:orders@pragprog.com">orders@pragprog.com</a>
Non-English Versions:	<a href="mailto:translations@pragprog.com">translations@pragprog.com</a>
Pragmatic Teaching:	<a href="mailto:academic@pragprog.com">academic@pragprog.com</a>
Author Proposals:	<a href="mailto:proposals@pragprog.com">proposals@pragprog.com</a>