Extracted from:

The ThoughtWorks Anthology

Essays on Software Technology and Innovation

Contents

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 decision 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 evolv-

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.

Contact Us

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