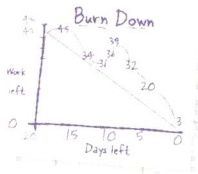
深入浅出软件开发（影印版）

Head First Software Development

Learn the real user story of how Mary satisfied her customers

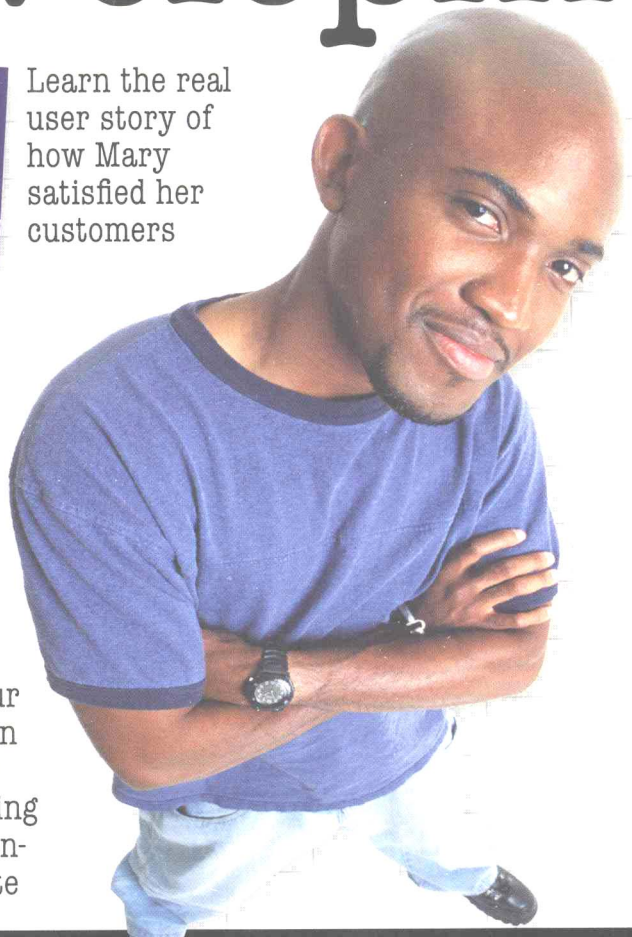
Score big by using velocity to figure out how fast your team can produce

Use test-driven development to avoid unsightly software disasters



Keep your project on schedule by tracking your burn-down rate

Master the techniques and tools of seasoned software developers



O'REILLY® 東南大學出版社

Dan Pilone & Russ Miles 著

深入浅出软件开发(影印版)

Head First Software Development

Dan Pilone
Russ Miles

O'REILLY®

Beijing • Cambridge • Köln • Paris • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc.授权东南大学出版社出版

南京　东南大学出版社

图书在版编目（CIP）数据

深入浅出软件开发：英文/（美）皮隆（Pilone, D.），（美）迈尔斯（Miles, R.）著. —影印本. —南京：东南大学出版社，2009.1

书名原文：Head First Software Development

ISBN 978-7-5641-1236-3

Ⅰ. 深… Ⅱ. ①皮… ②迈… Ⅲ. 软件开发－英文 Ⅳ. TP311.52

中国版本图书馆 CIP 数据核字（2008）第 104535 号

江苏省版权局著作权合同登记

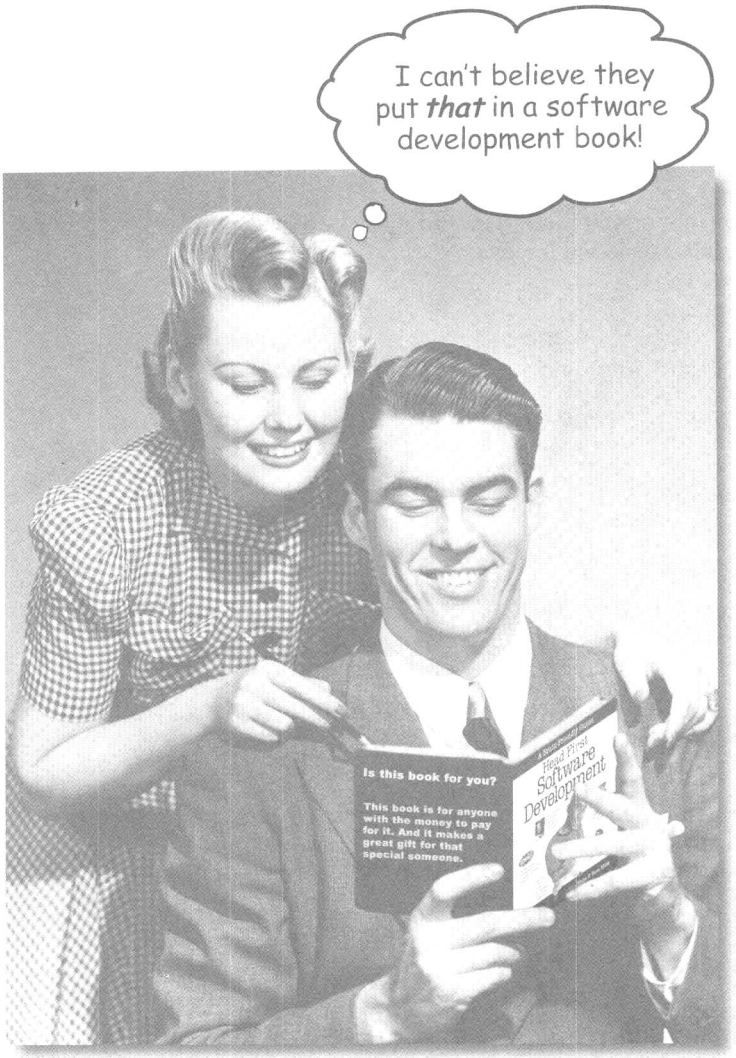
图字：10-2008-124 号

深入浅出软件开发（影印版）

出版发行：东南大学出版社
地　　址：南京四牌楼 2 号　　邮编：210096
出 版 人：江　汉
网　　址：http://press.seu.edu.cn
电子邮件：press@seu.edu.cn
印　　刷：扬中市印刷有限公司
开　　本：787 毫米 × 980 毫米　12 开本
印　　张：41.5 印张
字　　数：523 千字
版　　次：2009 年 1 月第 1 版
印　　次：2009 年 1 月第 1 次印刷
书　　号：ISBN 978-7-5641-1236-3/TP · 205
印　　数：1~3000 册
定　　价：88.00 元（册）

本社图书若有印装质量问题，请直接与读者服务部联系。电话（传真）：025-83792328

Intro



In this section we answer the burning question:
"So why DID they put that in a software development book?"

Who is this book for?

If you can answer "yes" to all of these:

1. Do you have access to a computer and **some background in programming**?
2. Do you want to **learn techniques for building and delivering great software?** Do you want to **understand** the principles behind iterations and test-driven development?
3. Do you prefer **stimulating dinner party conversation** to **dry**, **dull**, **academic lectures**?

We use Java in the book, but you can squint and pretend it's C#. No amount of squinting will make you think it's Perl, though.

this book is for you.

Who should probably back away from this book?

If you can answer "yes" to any of these:

1. **Are you completely new to Java?**

   (You don't need to be advanced, and if you know C++ or C# you'll understand the code examples just fine.)

2. Are you a kick-butt development manager looking for a ***reference* book**?

3. Are you **afraid to try something different**? Would you rather have a root canal than mix stripes with plaid? Do you believe that a technical book can't be serious if iterations are anthropomorphized?

this book is not for you.

[Note from marketing: this book is for anyone with a credit card.]

We know what you're thinking

"How can *this* be a serious software development book?"

"What's with all the graphics?"

"Can I actually *learn* it this way?"

We know what your *brain* is thinking

Your brain craves novelty. It's always searching, scanning, *waiting* for something unusual. It was built that way, and it helps you stay alive.

So what does your brain do with all the routine, ordinary, normal things you encounter? Everything it *can* to stop them from interfering with the brain's *real* job—recording things that *matter*. It doesn't bother saving the boring things; they never make it past the "this is obviously not important" filter.

How does your brain *know* what's important? Suppose you're out for a day hike and a tiger jumps in front of you, what happens inside your head and body?

Neurons fire. Emotions crank up. *Chemicals surge.*

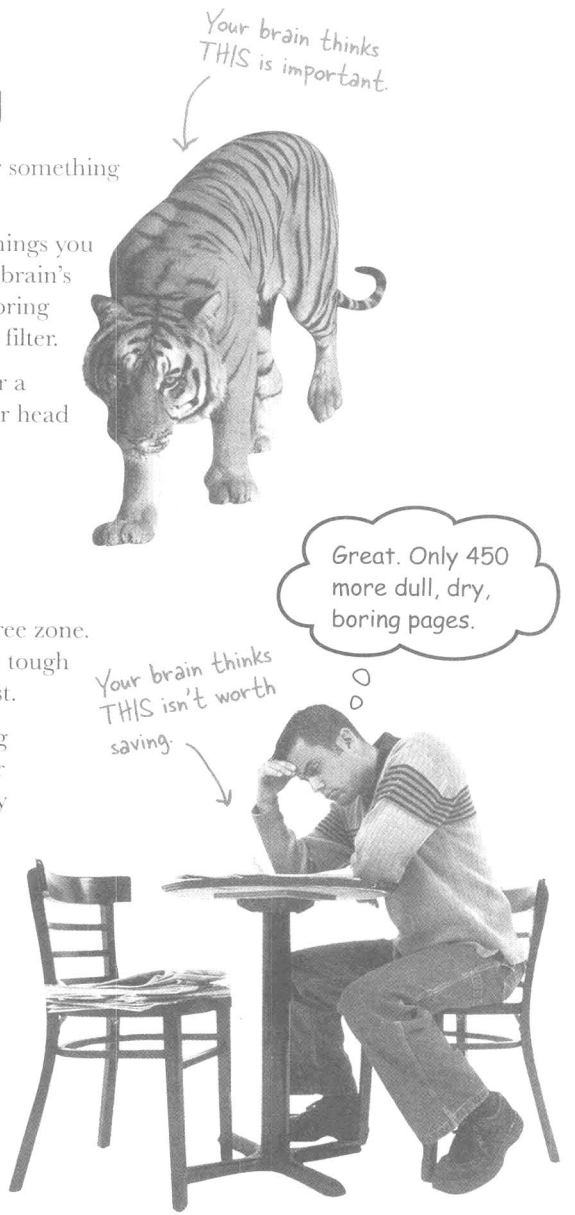
And that's how your brain knows...

This must be important! Don't forget it!

But imagine you're at home, or in a library. It's a safe, warm, tiger-free zone. You're studying. Getting ready for an exam. Or trying to learn some tough technical topic your boss thinks will take a week, ten days at the most.

Just one problem. Your brain's trying to do you a big favor. It's trying to make sure that this *obviously* non-important content doesn't clutter up scarce resources. Resources that are better spent storing the really *big* things. Like tigers. Like the danger of fire. Like the guy with the handle "BigDaddy" on MySpace probably isn't someone to meet with after 6 PM.

And there's no simple way to tell your brain, "Hey brain, thank you very much, but no matter how dull this book is, and how little I'm registering on the emotional Richter scale right now, I really *do* want you to keep this stuff around."



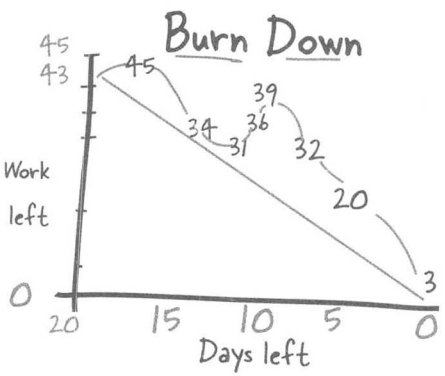
We think of a "Head First" reader as a learner.

So what does it take to *learn* something? First, you have to *get* it, then make sure you don't *forget* it. It's not about pushing facts into your head. Based on the latest research in cognitive science, neurobiology, and educational psychology, *learning* takes a lot more than text on a page. We know what turns your brain on.

Some of the Head First learning principles:

Make it visual. Images are far more memorable than words alone, and make learning much more effective (up to 89% improvement in recall and transfer studies). It also makes things more understandable. **Put the words within or near the graphics** they relate to, rather than on the bottom or on another page, and learners will be up to *twice* as likely to solve problems related to the content.

Use a conversational and personalized style. In recent studies, students performed up to 40% better on post-learning tests if the content spoke directly to the reader, using a first-person, conversational style rather than taking a formal tone. Tell stories instead of lecturing. Use casual language. Don't take yourself too seriously. Which would *you* pay more attention to: a stimulating dinner party companion, or a lecture?



Get the learner to think more deeply. In other words, unless you actively flex your neurons, nothing much happens in your head. A reader has to be motivated, engaged, curious, and inspired to solve problems, draw conclusions, and generate new knowledge. And for that, you need challenges, exercises, and thought-provoking questions, and activities that involve both sides of the brain and multiple senses.

Get—and keep—the reader's attention. We've all had the "I really want to learn this but I can't stay awake past page one" experience. Your brain pays attention to things that are out of the ordinary, interesting, strange, eye-catching, unexpected. Learning a new, tough, technical topic doesn't have to be boring. Your brain will learn much more quickly if it's not.

Touch their emotions. We now know that your ability to remember something is largely dependent on its emotional content. You remember what you care about. You remember when you *feel* something. No, we're not talking heart-wrenching stories about a boy and his dog. We're talking emotions like surprise, curiosity, fun, "what the...?", and the feeling of "I Rule!" that comes when you solve a puzzle, learn something everybody else thinks is hard, or realize you know something that "I'm more technical than thou" Bob from engineering *doesn't*.

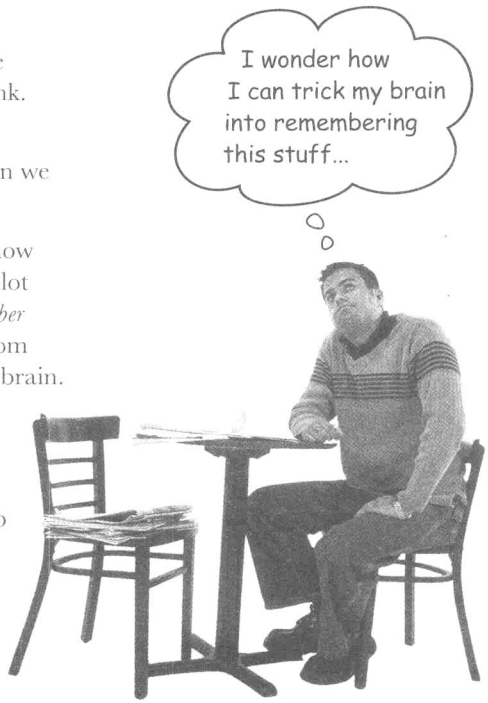
Metacognition: thinking about thinking

If you really want to learn, and you want to learn more quickly and more deeply, pay attention to how you pay attention. Think about how you think. Learn how you learn.

Most of us did not take courses on metacognition or learning theory when we were growing up. We were *expected* to learn, but rarely *taught* to learn.

But we assume that if you're holding this book, you really want to learn how to really develop great software. And you probably don't want to spend a lot of time. If you want to use what you read in this book, you need to *remember* what you read. And for that, you've got to *understand* it. To get the most from this book, or *any* book or learning experience, take responsibility for your brain. Your brain on *this* content.

The trick is to get your brain to see the new material you're learning as Really Important. Crucial to your well-being. As important as a tiger. Otherwise, you're in for a constant battle, with your brain doing its best to keep the new content from sticking.



So just how *DO* you get your brain to treat software development like it was a hungry tiger?

There's the slow, tedious way, or the faster, more effective way. The slow way is about sheer repetition. You obviously know that you *are* able to learn and remember even the dullest of topics if you keep pounding the same thing into your brain. With enough repetition, your brain says, "This doesn't *feel* important to him, but he keeps looking at the same thing *over* and *over* and *over*, so I suppose it must be."

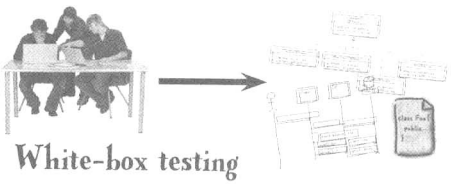
The faster way is to do ***anything that increases brain activity,*** especially different *types* of brain activity. The things on the previous page are a big part of the solution, and they're all things that have been proven to help your brain work in your favor. For example, studies show that putting words *within* the pictures they describe (as opposed to somewhere else in the page, like a caption or in the body text) causes your brain to try to makes sense of how the words and picture relate, and this causes more neurons to fire. More neurons firing = more chances for your brain to *get* that this is something worth paying attention to, and possibly recording.

A conversational style helps because people tend to pay more attention when they perceive that they're in a conversation, since they're expected to follow along and hold up their end. The amazing thing is, your brain doesn't necessarily *care* that the "conversation" is between you and a book! On the other hand, if the writing style is formal and dry, your brain perceives it the same way you experience being lectured to while sitting in a roomful of passive attendees. No need to stay awake.

But pictures and conversational style are just the beginning...

Here's what WE did:

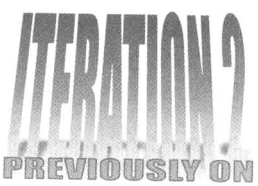
We used ***pictures***, because your brain is tuned for visuals, not text. As far as your brain's concerned, a picture really *is* worth a thousand words. And when text and pictures work together, we embedded the text *in* the pictures because your brain works more effectively when the text is *within* the thing the text refers to, as opposed to in a caption or buried in the text somewhere.



We used ***redundancy***, saying the same thing in *different* ways and with different media types, and *multiple senses*, to increase the chance that the content gets coded into more than one area of your brain.

We used concepts and pictures in ***unexpected*** ways because your brain is tuned for novelty, and we used pictures and ideas with at least *some* ***emotional*** *content*, because your brain is tuned to pay attention to the biochemistry of emotions. That which causes you to *feel* something is more likely to be remembered, even if that feeling is nothing more than a little ***humor***, ***surprise***, or ***interest.***

We used a personalized, ***conversational style***, because your brain is tuned to pay more attention when it believes you're in a conversation than if it thinks you're passively listening to a presentation. Your brain does this even when you're *reading*.



We included more than 80 ***activities***, because your brain is tuned to learn and remember more when you ***do*** things than when you *read* about things. And we made the exercises challenging-yet-do-able, because that's what most people prefer.

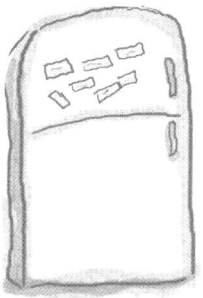
We used ***multiple learning styles***, because *you* might prefer step-by-step procedures, while someone else wants to understand the big picture first, and someone else just wants to see an example. But regardless of your own learning preference, *everyone* benefits from seeing the same content represented in multiple ways.

We include content for ***both sides of your brain***, because the more of your brain you engage, the more likely you are to learn and remember, and the longer you can stay focused. Since working one side of the brain often means giving the other side a chance to rest, you can be more productive at learning for a longer period of time.

And we included ***stories*** and exercises that present ***more than one point of view,*** because your brain is tuned to learn more deeply when it's forced to make evaluations and judgments.

We included ***challenges***, with exercises, and by asking ***questions*** that don't always have a straight answer, because your brain is tuned to learn and remember when it has to *work* at something. Think about it—you can't get your *body* in shape just by *watching* people at the gym. But we did our best to make sure that when you're working hard, it's on the *right* things. That ***you're not spending one extra dendrite*** processing a hard-to-understand example, or parsing difficult, jargon-laden, or overly terse text.

We used ***people***. In stories, examples, pictures, etc., because, well, because *you're* a person. And your brain pays more attention to *people* than it does to *things*.



Cut this out and stick it on your refrigerator.

Here's what YOU can do to bend your brain into submission

So, we did our part. The rest is up to you. These tips are a starting point; listen to your brain and figure out what works for you and what doesn't. Try new things.

1 Slow down. The more you understand, the less you have to memorize.

Don't just *read*. Stop and think. When the book asks you a question, don't just skip to the answer. Imagine that someone really *is* asking the question. The more deeply you force your brain to think, the better chance you have of learning and remembering.

2 Do the exercises. Write your own notes.

We put them in, but if we did them for you, that would be like having someone else do your workouts for you. And don't just *look* at the exercises. **Use a pencil.** There's plenty of evidence that physical activity *while* learning can increase the learning.

3 Read the "There are No Dumb Questions"

That means all of them. They're not optional sidebars—***they're part of the core content!*** Don't skip them.

4 Make this the last thing you read before bed. Or at least the last challenging thing.

Part of the learning (especially the transfer to long-term memory) happens *after* you put the book down. Your brain needs time on its own, to do more processing. If you put in something new during that processing time, some of what you just learned will be lost.

5 Drink water. Lots of it.

Your brain works best in a nice bath of fluid. Dehydration (which can happen before you ever feel thirsty) decreases cognitive function.

6 Talk about it. Out loud.

Speaking activates a different part of the brain. If you're trying to understand something, or increase your chance of remembering it later, say it out loud. Better still, try to explain it out loud to someone else. You'll learn more quickly, and you might uncover ideas you hadn't known were there when you were reading about it.

7 Listen to your brain.

Pay attention to whether your brain is getting overloaded. If you find yourself starting to skim the surface or forget what you just read, it's time for a break. Once you go past a certain point, you won't learn faster by trying to shove more in, and you might even hurt the process.

8 Feel something.

Your brain needs to know that this *matters*. Get involved with the stories. Make up your own captions for the photos. Groaning over a bad joke is *still* better than feeling nothing at all.

9 Write a lot of software!

There's only one way to learn to develop software: you have to **actually develop software**. And that's what you're going to do throughout this book. We're going to give you lots of requirements to capture, techniques to evaluate, and code to test and improve; every chapter has exercises that pose a problem for you to solve. Don't just skip over them—a lot of the learning happens when you solve the exercises. We included a solution to each exercise—don't be afraid to **peek at the solution** if you get stuck! (It's easy to get snagged on something small.) But try to solve the problem before you look at the solution.

Read Me

This is a learning experience, not a reference book. We deliberately stripped out everything that might get in the way of learning whatever it is we're working on at that point in the book. And the first time through, you need to begin at the beginning, because the book makes assumptions about what you've already seen and learned.

We assume you are familiar with object-oriented programming.

It would take an entire book to teach you object-oriented programming (like, say, **Head First OOA&D**). We chose to focus this book on software development principles rather than design or language basics. We picked Java for our examples because it's fairly common, and pretty self-documenting; but everything we talk about should apply whether you're using Java, C#, C++, or Visual Basic (or Ruby, or...) However, if you've never programmed using an object-oriented language, you may have some trouble following some of the code. In that case we'd strongly recommend you get familiar with one of those languages before attacking some of the later chapters in the book.

We don't cover every software development process out there.

There are tomes of information about different ways to write software. We don't try to cover every possible approach to developing code. Instead, we focus on techniques that we know work and fit well together to produce great software. Chapter 12 specifically talks about ways to tweak your process to account for unique things on your project.

The activities are NOT optional.

The exercises and activities are not add-ons; they're part of the core content of the book. Some of them are to help with memory, some are for understanding, and some will help you apply what you've learned. Some exercises are there just to make you think about **how** you would solve the problem. ***Don't skip the exercises.*** The crossword puzzles are the only thing you don't *have* to do, but they're good for giving your brain a chance to think about the words and terms you've been learning in a different context.

The redundancy is intentional and important.

One distinct difference in a Head First book is that we want you to *really* get it. And we want you to finish the book remembering what you've learned. Most reference books don't have retention and recall as a goal, but this book is about *learning*, so you'll see some of the same concepts come up more than once.

The examples are as lean as possible.

Our readers tell us that it's frustrating to wade through 200 lines of an example looking for the two lines they need to understand. Most examples in this book are shown within the smallest possible context, so that the part you're trying to learn is clear and simple. Don't expect all of the examples to be robust, or even complete—they are written specifically for learning, and aren't always fully functional.

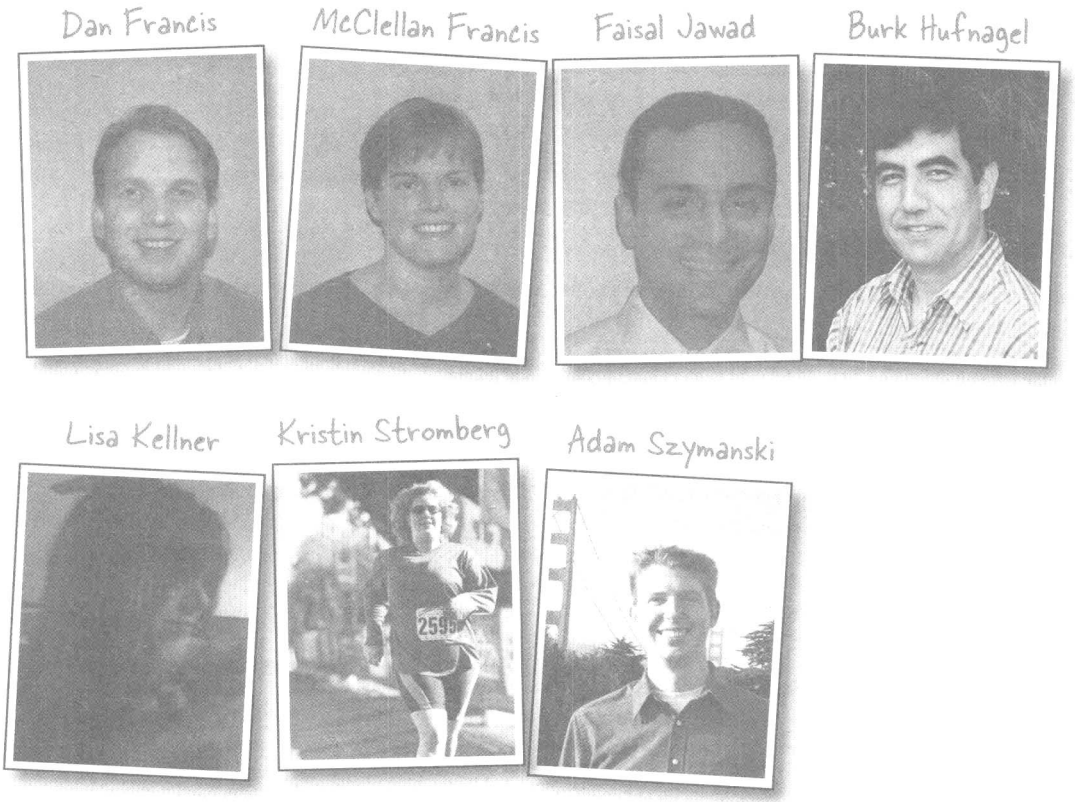
We've placed the full code for the projects on the Web so you can copy and paste them into your text editor. You'll find them at:

http://www.headfirstlabs.com/books/hfsd/

The Brain Power exercises don't have answers.

For some of them, there is no right answer, and for others, part of the learning experience of the Brain Power activities is for you to decide if and when your answers are right. In some of the Brain Power exercises, you will find hints to point you in the right direction.

The technical review team



Technical Reviewers:

This book wouldn't be anything like it is without our technical reviewers. They called us out when they disagreed with something, gave us "hurrah"s when something went right, and sent back lots of great commentary on things that worked and didn't work for them in the real world. Each of them brought a different perspective to the book, and we really appreciate that. For instance, **Dan Francis** and **McClellan Francis** made sure this book didn't turn into *Software Development for Java*.

We'd particularly like to call out **Faisal Jawad** for his thorough and supportive feedback (he started the "hurrahs"). **Burk Hufnagel** provided great suggestions on other approaches he'd used on projects and made one author's late night of updates a lot more fun with his suggestion to include, "Bad dev team. No biscuit."

Finally, we'd like to thank **Lisa Kellner** and **Kristin Stromberg** for their great work on readability and pacing. This book wouldn't be what it is without all of your input.

Acknowledgments

Our editor:

Don't let the picture fool you. **Brett** is one of the sharpest and most professional people we've ever worked with, and his contributions are on every page. Brett supported this book through every positive and negative review and spent quality time with us in Washington, D.C. to make this a success. More than once, he baited us into a good argument and then took notes while we went at it. This book is a result of his hard work and support and we really appreciate it.

Brett McLaughlin

The O'Reilly team:

Lou Barr

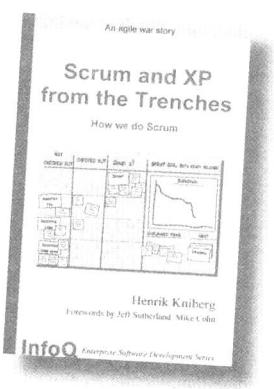
Lou Barr is the reason these pages look so "awesome." She's responsible for turning our vaguely worded "something that conveys this idea and looks cool" comments into pages that teach the material like no other book around.

Lou's favorite American-ism...

We'd also like to thank **Laurie Petrycki** for giving us the opportunity and making the tough decisions to get this book to where it is. We'd also like to thank **Catherine Nolan** and **Mary Treseler** for getting this book kicked off. Finally we'd like to thank **Caitrin McCullough**, **Sanders Kleinfeld**, **Keith McNamara**, and the rest of the O'Reilly production team for taking this book from the rough pages we sent in to a printed, high-class book with flawless grammar.

Scrum and XP from the Trenches:

We want to extend a special thanks to Henrik Kniberg for his book **Scrum and XP from the Trenches**. This book had significant influence on how we develop software and is the basis for some of the techniques we describe in this book. We're very grateful for the excellent work he's done.



Good book... highly recommended!

Our families:

No acknowledgments page would be complete without recognizing the contributions and sacrifices our families made for this book. Vinny, Nick, and Tracey have picked up the slack in the Pilone house for almost two years while this book came together. I can't convey how much I appreciate that and how your support and encouragement while "Daddy worked on his book" made this possible. Thank you.

A massive thank you also goes out to the Miles household, that's Corinne (the boss) and Frizbee, Fudge, Snuff, Scrappy, and Stripe (those are the pigs). At every step you guys have kept me going and I really can't tell you how much that means to me. Thanks!

Safari® Books Online

When you see a Safari® icon on the cover of your favorite technology book that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at `http://safari.oreilly.com`.

Table of Contents (Summary)

	Intro	xxv
1	great software development: *Pleasing your customer*	1
2	gathering requirements: *Knowing what the customer wants*	29
3	project planning: *Planning for success*	69
4	user stories and tasks: *Getting to the real work*	109
5	good-enough design: *Getting it done with great design*	149
6	version control: *Defensive development*	177
6.5	building your code: *Insert tab a into slot b...*	219
7	testing and continuous integration: *Things fall apart*	235
8	test-driven development: *Holding your code accountable*	275
9	ending an iteration: *It's all coming together...*	317
10	the next iteration: *If it ain't broke...you still better fix it*	349
11	bugs: *Squashing bugs like a pro*	383
12	the real world: *Having a process in life*	417

Table of Contents (the real thing)

Intro

Your brain on Software Development. You're sitting around trying to *learn* something, but your *brain* keeps telling you all that learning *isn't important*. Your brain's saying, "Better leave room for more important things, like which wild animals to avoid and whether naked rock-climbing is a bad idea." So how *do* you trick your brain into thinking that your life really depends on learning how to develop great software?

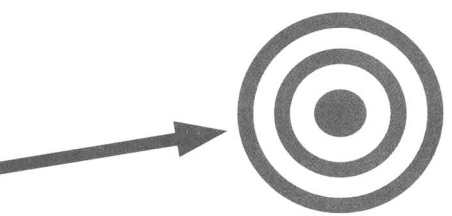
Who is this book for?	xxvi
We know what you're thinking	xxvii
Metacognition	xxix
Bend your brain into submission	xxxi
Read me	xxxii
The technical review team	xxxiv
Acknowledgments	xxxv

great software development

1 Pleasing your customer

If the customer's unhappy, *everyone's* unhappy!

Every great piece of software starts with a customer's big idea. It's your job as a professional software developer to **bring those ideas to life**. But taking a vague idea and turning it into working code—code that **satisfies your customer**—isn't so easy. In this chapter you'll learn how to avoid being a software development casualty by delivering software that is **needed**, **on-time,** and **on-budget**. Grab your laptop and let's set out on the road to shipping great software.



The Goal

Tom's Trails is going online	2
Most projects have two major concerns	3
The Big Bang approach to development	4
Flash forward: two weeks later	5
Big bang development usually ends up in a big MESS	6
Great software development is...	9
Getting to the goal with ITERATION	10
Each iteration is a mini-project	14
Each iteration is QUALITY software	14
The customer WILL change things up	20
It's up to you to make adjustments	20
But there are some BIG problems...	20
Iteration handles change automatically (well sort of)	22
Your software isn't complete until it's been RELEASED	25
Tools for your Software Development Toolbox	26