原 版 风 暴 系 列

PRENTICE HALL PTR

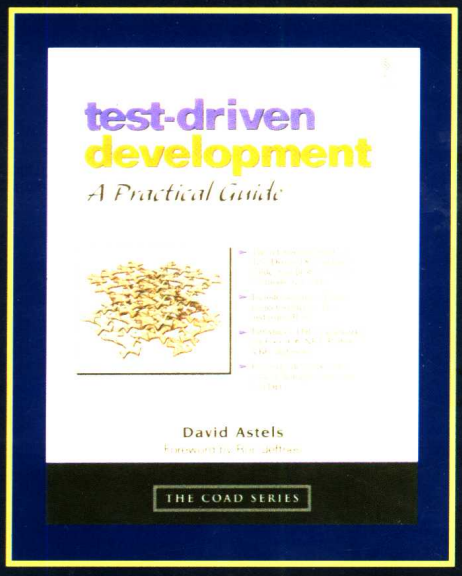
Test-Driven Development
A Practical Guide

测试驱动开发
实用指南
（影印版）

software development 14th annual product excellence award

本年度美国《软件开发》杂志震撼大奖得主

[美] David Astels 著
Ron Jeffries 序



- 最具实用性的测试驱动开发指南：问题、方案、代码
- 使用 Java 和 JUnit 测试框架从头开始创建项目
- 针对 C++、C#/.NET、VB6、Python 等的 TDD 框架
- 面向所有对 TDD 感兴趣的开发人员和项目经理

中国电力出版社
www.infopower.com.cn

原 版 风 暴 系 列

Test-Driven Development
A Practical Guide

测试驱动开发
实用指南

（影印版）

[美] David Astels 著
Ron Jeffries 序

中国电力出版社
www.infopower.com.cn

Test-Driven Development: A Practical Guide (ISBN 0-13-101649-0)

图书在版编目（CIP）数据
测试驱动开发：实用指南 /（美）艾斯特尔斯著. —影印本. —北京：中国电力出版社，2004
（原版风暴系列）
ISBN 7-5083-2193-6
Ⅰ.测... Ⅱ.艾... Ⅲ.软件开发－英文 Ⅳ.TP311.52
中国版本图书馆 CIP 数据核字（2004）第 028277 号

丛 书 名：原版风暴系列
书　　名：测试驱动开发：实用指南（影印版）
编　　著：（美）David Astels
责任编辑：姚贵胜
出版发行：中国电力出版社
　　地址：北京市三里河路6号　　邮政编码：100044
　　电话：（010）88515918　　传　　真：（010）88518169
印　　刷：北京丰源印刷厂
开　　本：787×1092　1/16　　印　　张：36.5
书　　号：ISBN 7-5083-2193-6
版　　次：2004 年 5 月北京第 1 版　　2004 年 5 月第 1 次印刷
定　　价：58.00 元

To my parents who,
though they may not have always
approved or understood,
let me try.

FOREWORD

My responsibility in writing this foreword is to help you decide whether to read this book. If you are interested in improving your programs and your programming skill, this book can help you.

Test-Driven Development is a practice that can make your programs better. If you're like me, using the techniques in Dave's book, you will find that your programs are more clear, that they come into being more easily, and that you'll have fewer defects than you used to.

I'm not saying that TDD is some kind of magic potion; quite the contrary. TDD isn't magic, it is something that you yourself do. By focusing attention on tests first, you'll be designing your program more from the viewpoint of the user. By doing the tests one at a time, you'll be creating a simple design that's focused exactly on the problem. As you work up all these little tests, you'll drive out most of the defects that otherwise slip into your code. Finally, by saving the tests, you make the program easier to maintain and improve as time goes on.

Dave's book is full of examples of Test-Driven Development. There's an extended example to show you how TDD works over a longer haul. There are small examples showing how to use most of the TDD-related tools that are available. There are even examples in most of the languages where TDD is used, though the book's main focus is on examples in Java. This is a book about practice, with real examples rather than dry theory.

But wait! There's more! Dave also gives a good introduction to refactoring, and to programming by intention. And he introduces Mock Objects, an advanced and powerful technique in testing and Test-Driven Development. Plus, he has a section on one of the tricky areas in TDD, creating GUIs in test-first fashion.

You'll also find a quick and valuable summary of eXtreme Programming, a look at Agile Modeling, and a comprehensive list of online resources relating to all the book's topics.

All these things are good and serve as reasons to buy this book. The core value of Dave's book, the real meat, is in the code. Test-Driven Development is a technique that we use as we program. No matter what design or modeling we have done before we begin programming, TDD helps us make the code better. I'm sure that it will help you, if you'll give this book, and what it teaches, a chance.

Test-Driven Development has made my programs better, and those of many other programmers as well. It's a technique that is worth adding to your bag of tricks. This book will help you improve as a programmer. That's why I'm recommending it.

Ron Jeffries
www.XProgramming.com
Pinckney, Michigan
18 December 2002

PREFACE

This isn't a book about testing.

This is a book about a way of programming that leads to simple, clear, robust code. Code that is easy to design, write, read, understand, extend, and maintain.

This is a book about thinking, designing, communicating, and programming. It's just a really nice side effect that we end up with a comprehensive[1] suite of tests.

This book explores Test-Driven Development, Test-First Programming, call it what you will: programming by writing the tests first, then writing the code needed to make the tests pass. Specifically, working in the smallest steps possible: write just enough of a test to fail, write just enough code to make it pass, refactor to clean up the mess you made getting the test to pass.

This book focuses on the Java programming language and uses Java examples throughout. It is assumed that the reader has at least an intermediate understanding of Java (and a working Java system if you want to try out the examples for yourself). Example code and other support material is available at my website[URL 54].

Even though the focus is on Java, Part IV looks at other prominent members of the xUnit family for several popular languages. This is done by taking the first task from Chapter 10 and recasting it in the various languages. This provides a good comparison of the different frameworks.

EXTREME PROGRAMMING

Test-Driven Development is a core part of the agile process formalized by Kent Beck called *eXtreme Programming* (XP). XP is probably the most agile of the agile processes, being extremely low overhead, and extremely low ceremony. However, it is extremely high discipline, very effective, and incredibly resilient to change.

That being said, you do not need to adopt XP in order to practice TDD and gain the benefit from it. TDD is worth doing on its own. The quality of your code will improve. Of course, if you are doing XP it's well worth it to get really good at TDD.

[1] How comprehensive depends on how good we become at it.

TDD is one of the main design tools that we have in XP.[2] As I mentioned earlier, the fact that we end up with a set of tests is a very pleasant by-product. Because we have those tests, we can have confidence we haven't inadvertently broken anything if the tests ran successfully before the change and after it. Conversely, if a test fails after we make a change we know exactly what broke and are in the best position to find the problem and fix it. The only thing that could have caused the failure was the change we made since the last time the tests ran clean.

All this means is that because the tests are there, we can safely use another of the XP practices: refactoring. As we will see in Chapter 2, refactoring is the process of making changes to the structure of code without changing its external behavior. The tests let you confirm that you haven't changed the behavior. This gives you the courage necessary to make (sometimes drastic) changes to working code. The result is that the code is cleaner, more extensible, more maintainable, and more understandable.

Appendix A talks a bit more about eXtreme Programming. For more exhaustive information, you can browse the bibliography and explore the online XP resources listed in Appendix C.

WHO SHOULD READ THIS BOOK?

Should you read this book? Helping you answer that question is why I wrote this preface. There was once an informal survey on the XP Yahoo Group as to the purpose that a preface should serve. The general opinion was that by reading the preface you should get a good idea of whether you should buy and read the book. I hope I've done a good job of it!

Read this book if you want to adopt eXtreme Programming. As stated earlier, being able to do TDD well is worth the time and effort it takes to get good at it. TDD is at the heart of XP, so doing TDD well makes the entire process that much more effective.

Read this book if you want to write code that is clearer, more robust, easier to extend, and as slim (as opposed to bloated) as possible.

Read this book if you know there must be a better way than spending weeks or months drawing pictures before writing a line of code.

Finally, read this book if you want to know how to make programming fun again.

In terms of what you should know before reading this book, it would help if you had at least an intermediate understanding of Java. Having a good background in another OO language or two (such as Smalltalk, C++, Python, or Ruby) will, however, enable you to get even more out of this book.

As this book goes to print there is one other TDD book available[9] (although I'm sure many will follow). I was aware of that book being written as I wrote much of this one, and it was always a partial goal to be complementary to it. From it you will get the philosophy and metaphysics of TDD, mixed with enough

[2] The other is refactoring.

pragmatics to make it real. If you are so inclined, I encourage you to read it first. The book you hold in your hands is, as the title says, a *practical* guide to doing TDD. It's focused on one language (not the best language, but arguably one that is very popular and well supported for TDD), and presents not only concepts and principles, but tools and techniques.

THE STRUCTURE OF THIS BOOK

This book is divided into four parts:

I Background In Part I we examine some topics that relate to the main body of material in the book (i.e., TDD in Java). We start with an introduction to TDD. This is followed by chapters on refactoring and programming by intention. These two techniques are also prominent in XP and are required and enabled by TDD.

II Tools and Techniques In Part II we take an in-depth look at various tools that are useful for practicing TDD with Java, and how to use them. We start with a tutorial introduction to JUnit, the defacto standard Java TDD framework. We continue by exploring some of the standard (i.e., included in the distribution) and nonstandard extensions to JUnit. Next, we explore some tools that support the use of JUnit and other tools that are completely independent of JUnit but work well with it. The final chapters in this part examine specific techniques or issues and the related tools.

III A Java Project: Test-Driven End to End This is a practical hands-on book. To that end, Part III (which makes up the bulk of the book) is built around the development of a real system, not a toy example. We work through this project test-first. Along the way we draw on material from the previous parts of the book.

IV xUnit Family Members JUnit is just one member of a large and growing family of programmer test frameworks. In Part IV we have a look at some of the other members of the family. We don't look at all of them, but we go over several for the more popular languages. So that we get a good comparison, we go through the same set of stories (i.e., requirements) for each. Specifically, these are the initial stories from the Java project. This lets us compare the various members with JUnit as well.

There are also four appendices:

A eXtreme Programming This appendix provides a very brief introduction to XP.

B Agile Modeling This appendix provides an introduction to and overview of Agile Modeling.

C Online Resources Throughout the book I refer to Web sites where you can find information as well as downloads. This appendix contains a categorized, annotated list of these sites.

D Answers to Exercises Many of the chapters in this book contain exercises for the reader. This appendix contains all exercises in the book, with answers.

CONVENTIONS USED IN THIS BOOK

I've adopted a handful of visual conventions which I've used throughout this book to make it easier for you, the reader, to differentiate between different sorts of information.

Source Code This book contains a large amount of source code. When one of more complete lines of code is being presented, it is indented and set in a sans-serif font, like this:

```
public int getAverageRating() {
  return totalRating / numberOfRatings;
}
```

When only part of a line is being presented, it is set in the same font, but kept in the body of the text. This often includes class names (Movie), methods (equals()), and constants (true, "a string", 42).

In general, when a method is referred to parameters are not included, but empty parentheses are, so that it is obvious that it is a method as opposed to some other type of identifier, for example: aMethod().

In blocks of code, package and import statements are generally left out.

Filesystem and console I/O Terms relating to the filesystem are set in a serif, monospaced font. This includes items like filenames (`filter.properties`) and commands and their output:

```
$ java \
> -classpath bin:/usr/local/java/lib/MockMaker.jar \
> mockmaker.MockMaker \
> com.saorsa.tddbook.samples.mockobjects.IntCalculator \
> >src/com/saorsa/tddbook/samples/mockobjects/MockIntCalculator.java
```

Tips and Sidebars I've used a couple of different callout mechanisms to highlight information that is important to take note of, or is interesting but doesn't fit in the body of the text for some reason.

Throughout the book there are small bits of wisdom that you may find especially useful. These are set apart the way this paragraph is.

A Sample Sidebar

This is an example of a very short sidebar; most are a half to full page in length.

I've used sidebars to separate short passages that are not directly related to the main body of text. Sidebars get placed as LaTeX sees fit, usually at the top of a page. There's one around here somewhere as an example.

Terminology I learned OO in the context of Smalltalk and I've used Smalltalk terminology from the beginning. If you're not familiar with Smalltalk, I include a few terms that I use, and how they map to Java and C++:

instance variable A variable whose scope is an object. Each object has a separate copy of this variable. (Java: *field* or *member variable*, C++: *data member*.)

class variable A variable whose scope is a class. All instances of the class share a single copy of the variable. (Java: *static field*, C++: *static data member*.)

method A functional member of a class. (Java: *method*, C++: *member function*.)

sending messages to an object A more abstract way to refer to calling an object's methods.

senders Methods that send a specific message, that is, call a specific method (commonly called *references* to a method).

ACKNOWLEDGEMENTS

A preface is not complete without acknowledging the other people that make a book possible. Being an author is like being at the peak of a pyramid... you are being supported (and your work made possible) in various ways by a multitude of other people. This is my chance to acknowledge and thank them... by name for the ones I'm aware of.

Kent Beck for making TDD and XP household words—at least in my household—and for his support of this book.

Miroslav Novak for first turning me on to this new way of programming that a bunch of smart people were talking about on something called a Wiki. Miroslav may be my junior in terms of time spent programming, but I've learned more from him than I sometimes care to admit.

Patrick Wilson-Welsh for several things: for always reminding me of the big picture when I got mired down in the details of the moment; for being the best

sounding board and copy editor that an author could ask for; and for having the courage to leave an established life in Washington, D.C. to move to small-town Canada to become my co-founder and first apprentice.

Dave Thomas of "The Pragmatic Programmers"[URL 55] for letting me use the LaTeX macros he wrote for the book "The Pragmatic Programmer"[25]. That book was inspiring in its layout and typesetting as well as catalytic in bringing about a turning point in my thoughts about programming.

Hand in hand with "The Pragmatic Programmer" went "Software Craftsmanship"[34] by Pete McBreen. I mean that literally,... I read them back-to-back. Pete provides a wonderful introduction to and discussion of software as a craft. A fabulous book, it was another contributing factor to my career-shaking epiphany (the third being XP). Thanks, Pete.

Peter Coad, to whom I owe a great debt for taking me under his wing in many ways and helping me to get this project off the ground. I have to thank him also for letting me charge ahead with a TDD edition of The Coad Letter[URL 61].

Paul Petralia, my acquisitions editor at Prentice Hall, and the fine crew that works with him. Thanks for letting us convince you that this book isn't about "Testing," and for believing in it wholeheartedly once we had accomplished that.

Craig Larman must be mentioned here for his encouragement, support, and advice. I still have great memories of spending a day with Craig at his home outside Dallas, discussing UML and Together[URL 34] and drinking homemade Chai.

And a big thanks to Ron Jeffries for writing the foreword for me, as well as being generally supportive of my XP-related endeavors, specifically (well, what comes to mind as I write this) this book, and the TDD Coad Letter. Also, for doing so much to bring XP so far.

Special thanks and a hearty acknowledgement to members of the TDD Yahoo! group that sent me their JUnit tips: Darren Hobbs, J. B. Rainsberger, and Derek Weber.

Very special thanks to those that contributed to the book by writing and letting me use material on subjects that they are the experts in, specifically (in order of appearance):

Mike Clark for the section on JUnitPerf,

Jens Uwe Pipka for the section on the Daedalos extensions,

Tim Bacon for the section on xmlUnit,

Mike Bowler for the section on the Gargoyle extensions,

Bryan Dollery for the section on IDEA,

James Newkirk for the chapter on NUnit,

Bob Payne for stepping in at the last minute with the chapter on PyUnit,

Kay Pentecost for the chapter on vbUnit, and

Scott Ambler for the appendix on agile modeling.

Thanks to all the folks in the XP community who gave me feedback (in no particular order): Kay Pentecost, Edmund Schweppe, Aldo Bergamini, Mike Clark, Francesco Cirillo, and my friends, colleagues, and past co-authors: Randy Miller and Miroslav Novak. As with all authors, I'm sure I've missed someone. Sorry about that.

I need to acknowledge and thank my reviewers as well: Alan Francis and William Wake.

And yes, as Kent Beck says in the preface of his TDD book[9], it is cliché to thank our families, but they heartily deserve it. To my wife, Kate, for saying "I'll clean up the kitchen. You go write." To my kids, Tasha and Jason, for being understanding when I had to write, and for thinking that it's so cool to have a Dad who writes books. Finally, to my youngest child, Leah, who is too young to notice what I'm doing but simply smiles when she sees me and gives me a hug when I pick her up.

This book was produced using a variety of open source software. All my computers run Redhat Linux. The manuscript was prepared using GNU Emacs, and typeset using LaTeX. Image manipulation was done with Gimp. The xdvi previewer was used extensively. The PDF version was created using dvips, and ps2pdf. Several packages were used with LaTeX, some off the shelf (lgrind, draftcopy, and fixme), several courtesy of Dave Thomas (for exercises, extended cross reference support, and url references), and several of my own (chapter heading quotes, story/task/test management, sidebars, and tips).

Contents