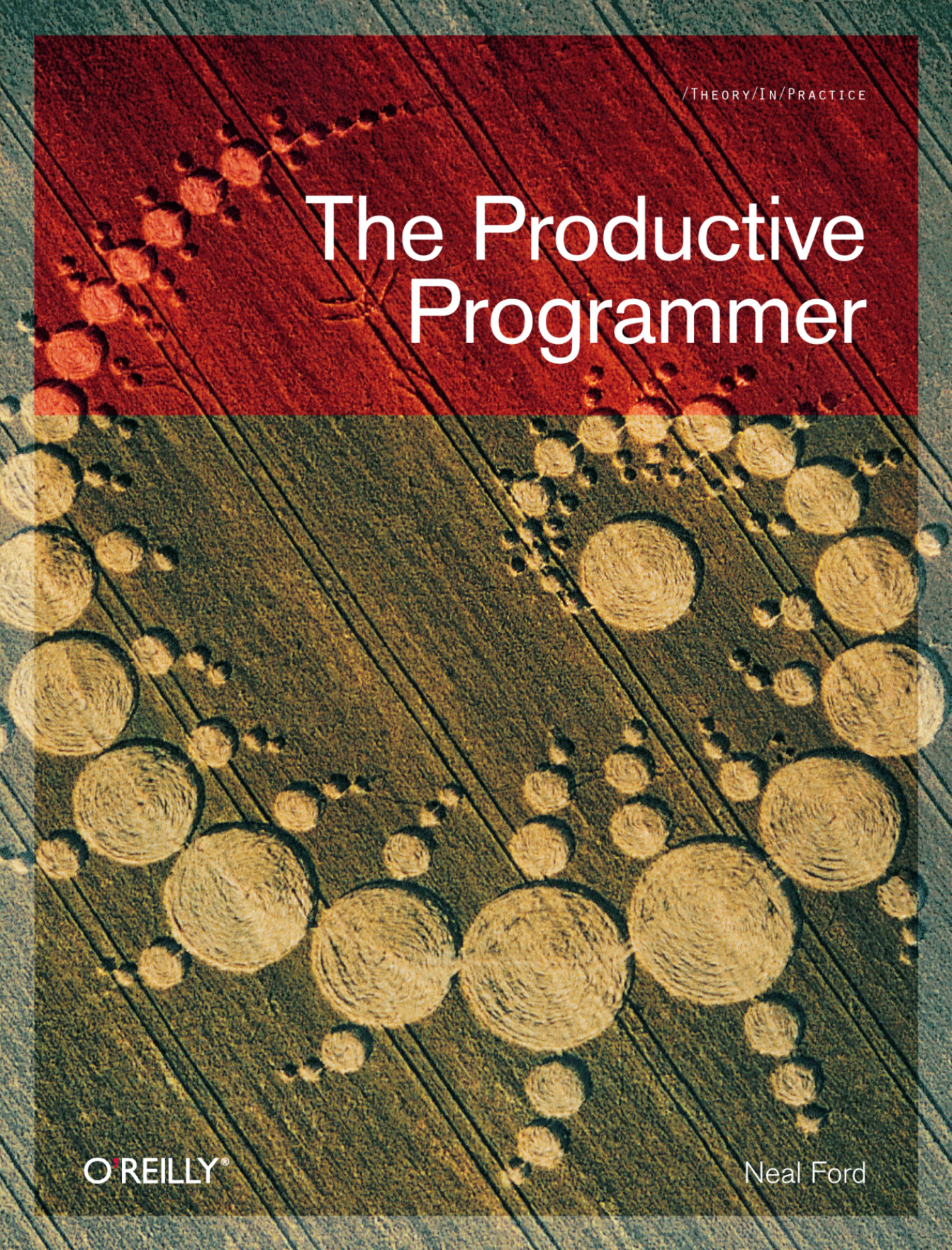
/Theory/In/Practice

The Productive Programmer



O'REILLY®

Neal Ford

The Productive Programmer

Neal Ford
foreword by David Bock

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

Copyrighted material

The Productive Programmer
by Neal Ford

Copyright © 2008 Neal Ford. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Mike Loukides
Production Editor: Loranah Dimant
Copyeditor: Emily Quill
Proofreader: Loranah Dimant

Indexer: Fred Brown
Cover Designer: Mark Paglietti
Interior Designer: David Futato
Illustrator: Robert Romano
Photographer: Candy Ford

Printing History:
July 2008: First Edition.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

This book uses RepKover™, a durable and flexible lay-flat binding.

ISBN: 978-0-596-51978-0

[C] [6/09]

1241550165

Copyrighted material

CONTENTS

	FOREWORD	vii
	PREFACE	ix
1	INTRODUCTION	1
	Why a Book on Programmer Productivity?	2
	What This Book Is About	3
	Where to Go Now?	5
Part One	MECHANICS	
2	ACCELERATION	9
	Launching Pad	10
	Accelerators	18
	Macros	33
	Summary	35
3	FOCUS	37
	Kill Distractions	38
	Search Trumps Navigation	40
	Find Hard Targets	42
	Use Rooted Views	44
	Use Sticky Attributes	46
	Use Project-Based Shortcuts	47
	Multiply Your Monitors	48
	Segregate Your Workspace with Virtual Desktops	48
	Summary	50
4	AUTOMATION	51
	Don't Reinvent Wheels	53
	Cache Stuff Locally	53
	Automate Your Interaction with Web Sites	54
	Interact with RSS Feeds	54
	Subvert Ant for Non-Build Tasks	56
	Subvert Rake for Common Tasks	57
	Subvert Selenium to Walk Web Pages	58
	Use Bash to Harvest Exception Counts	60
	Replace Batch Files with Windows Power Shell	61
	Use Mac OS X Automator to Delete Old Downloads	62
	Tame Command-Line Subversion	62
	Build a SQL Splitter in Ruby	64
	Justifying Automation	65

Copyrighted material

Don't Shave Yaks 67
Summary 68

5 CANONICALITY 69
DRY Version Control 70
Use a Canonical Build Machine 72
Indirection 73
Use Virtualization 80
DRY Impedance Mismatches 80
DRY Documentation 88
Summary 93

Part Two PRACTICE

6 TEST-DRIVEN DESIGN 97
Evolving Tests 99
Code Coverage 105

7 STATIC ANALYSIS 109
Byte Code Analysis 110
Source Analysis 112
Generate Metrics with Panopticode 113
Analysis for Dynamic Languages 116

8 GOOD CITIZENSHIP 119
Breaking Encapsulation 120
Constructors 121
Static Methods 121
Criminal Behavior 126

9 YAGNI 129

10 ANCIENT PHILOSOPHERS 135
Aristotle's Essential and Accidental Properties 136
Occam's Razor 137
The Law of Demeter 140
Software Lore 141

11 QUESTION AUTHORITY 143
Angry Monkeys 144
Fluent Interfaces 145
Anti-Objects 147

12 META-PROGRAMMING 149
Java and Reflection 150
Testing Java with Groovy 151
Writing Fluent Interfaces 152
Whither Meta-Programming? 154

13 COMPOSED METHOD AND SLAP 155
Composed Method in Action 156

Copyrighted material

SLAP 160

14 POLYGLOT PROGRAMMING 165
How Did We Get Here? And Where Exactly Is Here? 166
Where Are We Going? And How Do We Get There? 169
Ola's Pyramid 173

15 FIND THE PERFECT TOOLS 175
The Quest for the Perfect Editor 176
The Candidates 179
Choosing the Right Tool for the Job 180
Un-Choosing the Wrong Tools 186

16 CONCLUSION: CARRYING ON THE CONVERSATION 189

APPENDIX: BUILDING BLOCKS 191

INDEX 199

Copyrighted material

Copyrighted material

FOREWORD

The individual productivity of programmers varies widely in our industry. What most of us might be able to get done in a week, some are able to get done in a day. Why is that? The short answer concerns mastery of the tools developers have at their disposal. The long answer is about the real *awareness* of the tools' capabilities and mastery of the thought process for using them. The truth lies somewhere between a methodology and a philosophy, and that is what Neal captures in this book.

The seeds of this book were planted in the fall of 2005, on a ride back to the airport. Neal asked me, "Do you think the world needs another book on regular expressions?" From there, the conversation turned to topics of books we wished existed. I thought back to a point in my career where I feel I made the leap from merely good to very productive, and how and why that happened. I said, "I don't know what the title of the book is, but the subtitle would be 'using the command line as an integrated development environment.'" At the time I credited my increased productivity to the acceleration I experienced using the bash shell, but it was more than that—it was my increasing familiarity with that tool as I stopped having to struggle to do things and could just get them done. We spent some time discussing that hyperproductivity and how to bottle it. Several years, untold conversations, and a series of lectures later, Neal has produced a definitive work on the subject.

In his book *Programming Perl* (O'Reilly), Larry Wall describes the three virtues of a programmer as "laziness, impatience, and hubris." Laziness, because you will expend effort to reduce the amount of overall work necessary. Impatience, because it will anger you if you are wasting time doing something the computer could do faster for you. And hubris, because excessive pride will make you write programs that other people won't say bad things about. This book doesn't use any of those words (and I used *grep* to check), but as you read on, you will find this sentiment echoed and expanded in this content.

Copyrighted material

There are several books that have had a great influence on my career, changing the way I see the world. I wish I had this book in hand 10 years ago; I'm sure it will have a profound influence on those who read it.

—David Bock
Principal Consultant
CodeSherpas

Copyrighted material

PREFACE

Copyrighted material

Many years ago, I taught training classes for experienced developers who were learning new technologies (like Java). The disparity between the productivity of the students always struck me: some were orders of magnitude more effective. And I don't mean in the tool they were using: I mean in their general interaction with the computer. I used to make a joke to a few of my colleagues that some of the people in the class weren't running their computers, they were walking them. Following a logical conclusion, that made me question my own productivity. Am I getting the most efficient use out of the computer I'm running (or walking)?

Fast-forward years later, and David Bock and I got into a conversation about this very thing. Many of our younger coworkers never really used command-line tools, and didn't understand how they could possibly offer more productivity than the elaborate IDEs of today. As David recounts in the foreword to this book, we chatted about this and decided to write a book about using the command line more effectively. We contacted a publisher, and started gathering all the command-line voodoo we could find from friends and coworkers.

Then, several things happened. David started his own consulting company, and he and his wife had their first children: triplets! Well, David now clearly has more on his hands than he can handle. At the same time, I was coming to the conclusion that a book purely about command-line tricks would be perhaps the most boring book ever written. At about that time, I was working on a project in Bangalore, and my pair-programmer partner, Mujir, was talking about code patterns and how to identify them. It hit me like a ton of bricks. I had been seeing patterns in all the recipes I'd been gathering. Instead of a massive collection of command-line tricks, the conversation should be about *identifying* what makes developers more productive. That's what you hold in your hands right now.

Who This Book Is For

This isn't a book for end users who want to use their computers more effectively. It's a book about *programmer* productivity, which means I can make a lot of assumptions about the audience. Developers are the ultimate power users, so I don't spend a lot of time on basic stuff. A tech-savvy user should certainly learn something (especially in Part I), but the target remains developers.

There is no explicit order to this book, so feel free to wander around as you like or read it front to back. The only connections between the topics appear in unexpected ways, so reading it front to back may have a slight advantage, but not enough to suggest that's the only way to consume this book.

Copyrighted material

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*The Productive Programmer* by Neal Ford. Copyright 2008 Neal Ford, 978-0-596-51978-0."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707 829-0104 (fax)

Copyrighted material

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/9780596519780

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com

Safari® Enabled

When you see a Safari® Enabled 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*.

Acknowledgments

This is the only part of the book my non-techy friends will read, so I'd better make it good. My entire life-support system has helped me greatly in this long, drawn-out book process. First, my family, especially my mom Hazel and dad Geary, but also my entire extended family, including my stepmother Sherrie and my stepdad Lloyd. The No Fluff, Just Stuff speakers, attendees, and the organizer Jay Zimmerman have helped me vet this material over many months, and the speakers in particular make the ridiculous amount of travel worthwhile. A special thanks goes to my ThoughtWorks colleagues: a group of people with whom I feel extraordinarily privileged to work. I've never before seen a company as committed to revolutionizing the way people write software, with such highly intelligent, passionate, dedicated, selfless people. I attribute at least some of this to the extraordinary Roy Singham, the founder of ThoughtWorks, and upon whom I have a bit of a man-crush, I think. Thanks to all my neighbors (both the non-garage and honorary garage ones), who don't know or care about any of this technology stuff, especially Kitty Lee, Diane and Jamie Coll, Betty Smith, and all the other current and former Executive Park neighbors (and yes that includes you Margie). Special thanks to my friends that now extend around the globe: Masoud Kamali, Frank Stepan, Sebastian Meyen, and the rest of the S&S crew. And, of course, the guys I see only in other countries, like Michael Li, and, even though they live only five miles away, Terry Dietzler and his wife Stacy, whose schedules far too rarely line up with mine. Thanks (even though they can't read this) to Isabella, Winston, and Parker, who don't care about technology but really

Copyrighted material

care about attention (on their terms, of course). A thanks to my friend Chuck, whose increasingly rare visits still manage to lighten my day. And, saving the most important for last, my wonderful wife Candy. All my speaker friends claim that she's a saint for allowing me to gallivant around the world, speaking about and writing software. She has graciously indulged my all-encompassing career because she knows I love it, but not as much as her. She's patiently waiting around until I retire or tire of all this, and I can spend all my time with her.

A special thanks goes out to the technical reviewers for this book. Without their hard work and dedication, this book would suffer lots of silly mistakes and confusing explanations. Thanks to Greg Ostravich (who has reviewed every book of mine for the last few years and gotten no recognition, unfortunately), Venkat Subramaniam, David Bock, Nathaniel Schutta, and Matthew McCullough.

Copyrighted material

Copyrighted material

CHAPTER ONE

Introduction

Copyrighted material

PRODUCTIVITY IS DEFINED AS THE AMOUNT OF USEFUL WORK PERFORMED OVER TIME. Someone who is more productive performs more effective work in a given time interval than someone less productive. This book is all about how to become more productive as you go about the tasks required to develop software. It is language and operating system agnostic: I provide tips in a variety of languages, and across three major operating systems: Windows (in various flavors), Mac OS X, and *-nix (Unix and Linux alternatives).

This book is about individual programmer productivity, not group productivity. To that end, I don't talk about methodology (well, maybe a little here and there, but always on the periphery). I also don't discuss productivity gains that affect the whole team. My mission is to allow individual programmers the tools and philosophies to get more useful work done per unit of time.

Why a Book on Programmer Productivity?

I work for ThoughtWorks, an international consulting company of about 1,000 employees spread across 6 countries. Because we are traveling consultants (especially in the U.S.), we are a demographically very young company. It came to pass at one of our company outings (where beverages were served) that I starting chatting with one of the People people. She asked me how old I was and I told her. Then, she gave me an off-handed compliment(?): "Wow, you're old enough to add diversity to the company!" That sparked some thoughts. I've been developing software for many years (cue the maudlin "Back in my day, we had kerosene-powered computers…"). During that time, I've observed an interesting phenomenon: developers are getting less efficient, not more. Back in ancient times (a couple of decades in computer time), *running* a computer was a difficult job, much less programming the thing. You had to be a really clever developer to get anything useful from the beastly machine. This crucible forged Really Smart Guys who developed all sorts of efficient ways to interact with the intractable computers of their age.

Slowly, due to the hard work of programmers, computers became easier to use. This innovation was really to stop users from complaining so much. The Really Smart Guys congratulated themselves (as all programmers do when they can get a user to pipe down). Then a funny thing happened: a whole generation of developers came along who no longer needed clever tricks and devious intelligence to get computers to do what they wanted. The developers, like the end users, basked in the easier-to-use computers. So, what's wrong with that? After all, productivity is a good thing, right?

It depends. What is productive for a user (nice graphical user interface, mice, pull-down menus, etc.) can actually be a hindrance to someone trying to get the best performance from a computer. "Easy to use" and "efficient" overlap infrequently. Developers who grew up using graphical user interfaces (OK, I'll just go ahead and say it: Windows) don't know many of the cool, efficient tricks of the trade of the Really Smart Guys of yesteryear. Developers today are not *running* their computers, they are *walking* them. I'm going to try to fix that.

Copyrighted material

Address Completion in Browsers

Here's a quick example: how many web sites do you visit in a day? Most of them start with "www." and end with ".com". A little-known shortcut exists for all modern browsers: *address completion*. Address completion uses a hotkey combination to automatically add "www." to the beginning and ".com" to the end of the string you type in the browser's address bar. Different browsers support slightly different syntax. (Note that this is different from letting the browser automatically supply the prefix and suffix. All the modern browsers do that too.) The difference is one of efficiency. To autocomplete the prefix and suffix, the browser goes out to the network and looks for a site with the "bare" name. If it doesn't find one, it tries it with the prefix and suffix, entailing another trip out to the network. With a fast connection, you may not even notice the lag, but you're slowing down the whole Internet with all those false hits!

Internet Explorer

Internet Explorer (IE) makes it easier to type in addresses that contain standard prefixes and suffixes. Use the keys Ctrl-Enter to add "www." to the front of an address and ".com" to the end.

Firefox

The same Internet Explorer shortcut works for the Windows version of Firefox as well. For Macintosh, Apple-Enter does the same thing. Firefox goes one better: for all the platforms it supports, Alt-Enter places a ".org" at the end.

Firefox has other handy shortcut keys that no one seems to leverage. To go directly to a tab, you can use Ctrl + <TAB-NUMBER> in Windows or Apple + <TAB-NUMBER> in OS X.

OK, this shortcut is worth a measly eight keystrokes per web page. But think of the number of web pages you visit every day, and those eight characters per page start to add up. This is an example of the principle of *acceleration*, defined in Chapter 2.

But saving eight keystrokes per web page isn't the point of this example. I conducted an informal poll of all the developers I know and learned that less than 20 percent of them knew this shortcut. These folks are hardcore computer experts, yet they weren't taking advantage of even the simplest productivity gains. My mission is to rectify that.

What This Book Is About

The Productive Programmer is divided into two parts. The first discusses the *mechanics* of productivity, and the tools and their uses that make you more productive as you go through the physical activities of developing software. The second part discusses the *practice* of productivity, and how you can leverage your knowledge and the knowledge of others to produce better software more quickly. In both sections, you will likely run into some things you already know, as well as things you have never thought of before.

Copyrighted material

Part I: Mechanics (The Productivity Principles)

You can treat this book as a recipe book for command-line and other productivity tips and still reap benefits. But if you understand *why* something increases productivity, you can recognize it all around you. Creating patterns to describe something creates *nomenclature*: once you have a name for something, it's easier to recognize when you see it again. One of the goals of this book is to define a set of productivity principles to help you define your own productivity techniques. Like all patterns, it becomes easier to identify them once they have names. Knowing why something speeds you up allows you to more quickly identify other things that will help you work faster.

This is not just a book on how to use computers more effectively (although that is a side effect). It is focused on *programmer* productivity. To this end, I don't cover many things that are obvious to casual or even power users (although, as the exception that proves the rule, the earlier section "Address Completion in Browsers" does show an obvious tip). Programmers represent a unique subsection of computer users. We should be able to bend computers to our will more effectively than anyone else because we understand the most about how they really work. Mostly, this book is about things you can do with and to a computer to make your job easier, faster, and more efficient. However, I also discuss some low-hanging fruit that can make you more productive.

Part I covers every productivity tip I could invent, harvest, bully out of my friends, or read about. Originally, I aimed to create the world's most awesome collection of productivity recipes. I don't know if that happened, but you will still find a pretty impressive collection of recipes here.

As I started collating all these cool productivity tips, I noticed patterns emerging. Looking over these techniques, I started formulating categories of productivity for programmers. Eventually, I created the *Principles of Programmer Productivity* because, frankly, I couldn't think of a more pretentious name. These principles are *acceleration*, *focus*, *automation*, and *canonicality*. They describe the practices that allow programmers to become more productive.

Chapter 2, *Acceleration*, describes becoming more productive by speeding something up. Obviously, if one person is faster at a particular task, that person is more productive at that task than someone who goes more slowly doing the same thing. Great examples of the acceleration principle are the numerous keyboard shortcuts that appear throughout the book. Acceleration encompasses things like launching applications, managing clipboards, and searching and navigation.

Chapter 3, *Focus*, describes how to achieve the state of super-productivity, using both tools and environmental factors. It discusses ways to reduce the clutter of your environment (both physical and virtual), how to search efficiently, and how to avoid distractions.

Copyrighted material

Getting the computer to perform extra work for you obviously makes you more productive. Chapter 4, *Automation*, describes coercing your computer to do more work for you. Many of the tasks you perform every day can (and should) be automated. This chapter has examples and strategies to put your computer to work.

Canonicality is really just a fancy term for the application of the DRY (Don't Repeat Yourself) principle, first espoused in *The Pragmatic Programmer* (Addison-Wesley) by Andy Hunt and Dave Thomas. The DRY principle advises programmers to find places where information is duplicated and create a single source for this information. *The Pragmatic Programmer* eloquently describes this principle, and in Chapter 5, *Canonicality*, I show concrete examples of applying it.

Part II: Practice (Philosophy)

I've worked as a consultant for most of my many seasoned years as a developer. Consultants have advantages over developers who work on the same code base year after year. We get to see lots of different projects and lots of different approaches. Of course, we see our share of train wrecks as well (rarely do consultants get called in to "fix" healthy projects). We get to see the broad spectrum of software development: building things from the start, advising in the middle, and rescuing what's badly broken. Over time, even the least observant person can get a feel for what works and what doesn't.

Part II is the distillation of the things I've seen that either make developers more productive or detract from their productivity. I've bundled them together in a more or less random order (although you may be surprised by how often the same ideas come up in different guises). This isn't meant to be the ultimate compendium of things that make developers productive; rather, it is the list of things I've observed, which is just a small subset of the possibilities.

Where to Go Now?

The two parts of this book stand alone, so you can read them in any order; however, Part II is a tad more narrative, and unexpected connections may pop up. Still, most of the material in it is nonsequential: you can read it in any order you like.

One note of warning. If you aren't comfortable with basic command-line stuff (pipes, redirection, etc.), you should make a quick visit to Appendix A. It covers getting an environment set up that is suitable for using many of the tricks and techniques discussed in Part I. It's pretty painless, I promise.

Copyrighted material

Copyrighted material

PART I

Mechanics

Part I, *Mechanics*, deals with (yup, you guessed it) the mechanics of productivity. Many of these tools aren't necessarily developer tools, but rather tools that could be of help to any sophisticated power user. Of course, developers should be the ultimate power users, taking full advantage of virtually all the tool categories listed in this part of the book.

Copyrighted material

Copyrighted material

CHAPTER TWO

Acceleration

Copyrighted material

Using a computer requires a fair amount of ritual and ceremony. You have to boot it up, you have to know how to launch applications, and you must understand the interaction model, which can differ between applications. The less you interact with your computer, the faster you can go. In other words, eliminating ceremony allows you more time to get to the essence of the problem. Time you spend digging through a long filesystem hierarchy to find something is time you could be using to be more productive. Computers are tools, and the more time you spend on the care and feeding of the tool, the less work you get done. The science fiction author Douglas Adams had a great quote: "We are stuck with technology when what we really want is just stuff that works."

> NOTE
> **Concentrate on essence, not ceremony.**

This chapter is all about figuring out ways to accelerate your interaction with your computer, whether it's launching applications more quickly, finding files faster, or using the mouse less.

Launching Pad

Take a look at your computer's list of applications. If you are using Windows, click on Start and choose Programs. How many columns do you have? Two? Three? Four!? As hard drives have gotten bigger and the kinds of applications (and therefore the tools we must use) have gotten more complex, the number of applications we use has exploded. Of course, with commonplace 100 GB drives, we can pack lots of stuff into our systems. But volume exacts a price.

> NOTE
> **The usefulness of an application list is inversely proportional to its length.**

The longer the list, the less useful it becomes. With three columns in Windows or a dock that squeezes items to microscopic size in Mac OS X, it's getting harder and harder to find what we need. This hits developers particularly hard because we have lots of *occasional applications*: special-purpose tools that we may run only one day a month, but that we desperately need when that day arrives.

Launchers

Launchers are applications that allow you to type the first part of the name of an application (or document) to launch it. Most of the time, this is a more efficient way to launch applications.

> NOTE
> **Eye candy looks good but isn't nutritious.**

Copyrighted material

If you know the name of the thing you are after (like the name of the application), why not just tell your computer what you want, rather than sort through a massive list or search for it in a sea of icons? Launchers cut through the graphical eye candy and drill precisely and quickly to the thing you need.

All the major operating systems have open source and free launchers that allow you to type the name (or a portion of the name) of the application you want to launch. A few that are worth trying are Launchy,* Colibri,† and Enso.‡ Both Launchy and Colibri are open source and therefore free, and they both allow you to open a small window and start typing the name of your application, which will pop up on a list. Launchy is currently the most popular of the open source launchers. Colibri is attempting to reproduce a Mac OS X utility called Quicksilver (discussed in the upcoming section "Mac OS X").

Enso is a launcher with some interesting added features. It is also free (but not open source), created by Humanized, the company founded by Jef Raskin, one of the early user interface designers for the Mac. Enso encapsulates many of his (sometimes slightly radical) user interface views, but it is quite effective. For example, one of the ideas that Raskin promotes is the idea of Quasimode keys, which act like the Shift key (in other words, changing the mode of the keyboard when held down). Enso takes over the pretty worthless Caps Lock key and uses it to launch applications and perform other tasks. You hold the Caps Lock key down and start typing a command, like *OPEN FIREFOX*, and it will open Firefox. Of course, that is cumbersome to type, so another Enso command is *LEARN AS FF FIREFOX*, which teaches Enso that the *FF* command launches Firefox. Enso does more than just launching. If you have a math expression like 4+3 in a document, you can highlight it and invoke the *CALCULATE* command, and Enso will replace your highlighted text with the value of the calculation. Enso is worth trying to see if Raskin's views on launching mesh with yours.

If you are using Windows Vista, it includes the launching part of these launcher applications. When you invoke Start, the bottom of the ensuing menu has a search field, allowing you to type the name of the application you want, using incremental searching. But it has one disadvantage (perhaps a bug) that the launchers mentioned above don't have: if you type something that doesn't exist, Windows Vista takes a very long time to come back and tell you that it can't find it. Your machine is about as useful as a brick while this is happening. Hopefully, this is just a quirk in the current version and will be fixed soon.

* Download at *http://www.launchy.net.*

† Download at *http://colibri.leetspeak.org.*

‡ Download at *http://www.humanized.com.*

Copyrighted material

FIGURE 2-1. Custom launcher window

Creating a Windows Launching Pad

You can easily take advantage of the folder infrastructure in Windows to create your own launching pad. Create a folder under the Start button that contains shortcuts to the applications you use on a daily basis. You might call this folder "jump" and use "j" as its shortcut key, so you can access it by typing Windows-J. An example of just such a "jump" window appears in Figure 2-1. Notice that each of the menu items includes a single-letter prefix that is unique within the folder, which facilitates launching applications fast. Every application in the launch folder is just two keystrokes away: Windows-J[<unique letter>] launches the application.

This is just a directory, so you can nest other directories inside it to create your own mini-hierarchy of applications. For most of my workhorse machines, 26 entries isn't enough, so I tend to create a *dev* launch folder that contains all the development tools. I'm willfully trading one hierarchy for another, but with a big difference: I have complete control over this organization, unlike the Programs group in Windows. I regularly reorganize this folder as some applications fall out of favor and new ones take their place.

Creating the launching folder is very simple, but it depends on the flavor of Start button you have. Windows XP and Vista support two kinds of "Start" configurations: "classic" (the style from Windows 95 through 2000) and "modern" (Windows XP and Vista). For "classic" Windows, creating the launch folder is extremely simple. Right-click on the Start button, choose either Open (if you just want to add a launch menu for the currently logged-in user) or Open All Users (to change it for everyone). This opens the underlying folder that controls the contents of the "Start" menu, where you can add shortcuts to your heart's content. Alternatively, you can go to where the Start menu lives, under the current user's Documents and Settings directory structure. An easy way to fill up your launch menu with just the stuff

Copyrighted material

you need all the time is to select them from the massive Programs menu and right-drag them into your launch folder, creating a copy of the shortcut.

If you have the "modern" Windows Start menu, creating a launch menu is tougher, but still possible. You can create the launch folder in the same directory as mentioned earlier, but for some odd reason, it no longer appears right when you hit the Windows key; it now appears only after you expand the Programs group. This is a major annoyance because now our accelerated way to launch stuff takes an extra keystroke; however, there is a way around this problem. If you create your *jump* folder on the desktop and drag-drop it onto the Start menu, it will create a folder that shows up right away. The only remaining headache with the "modern" version of the Start menu concerns the hotkey. In the "modern" menu, different applications move in and out, depending on usage (Windows randomizes your carefully memorized paths to get to things quickly). So, if you use the "modern" Start menu, you should choose a first character for your launch menu that won't conflict, like the ~ or (keys.

Because it is such as hassle, I tend to just use the "classic" version of the Start button. You can change from "modern" to "classic" in either Windows XP or Vista via the properties of the taskbar, as shown in Figure 2-2.

RE-WIRING SPECIAL FOLDERS IN WINDOWS

Microsoft issues (but doesn't support) a collection of utilities known as PowerToys,[§] including Tweak UI, which lets you make changes to your Windows registry through a graphical interface. My Documents normally resides at the knuckle-bending location of *c:\Documents and Settings\<your login name>\My Documents* (mercifully changed to just *Documents*, directly off the root in Windows Vista). Tweak UI allows you to change the default location of My Documents so that you can move it to a more sane location like *c:\Documents* (where Windows Vista puts your documents by default).

Be careful, though: if you are going to move My Documents, you should do it early in the life of your install of the operating system. Lots of Windows applications rely on artifacts in that directory, and you will subsequently break lots of applications if you do this on a well-established Windows machine.

If you don't want to go quite this far, you can select a folder (like My Documents), right-click to get the properties dialog, and tell Windows to relocate it. It will copy all your My Documents files to the new location. You can also use the ancient *subst* command (which allows you to substitute one folder for another), but it is known to break lots of applications, so use it with caution. Junction, a utility that lets you truly substitute one directory for another, works better if you are using the NTFS filesystem. See "Indirection" in Chapter 5 for more details.

§ Download at *http://www.microsoft.com/windowsxp/downloads/powertoys/xppowertoys.mspx*.

Copyrighted material

FIGURE 2-2. Changing back to the classic menu

Windows does have a quick and easy mechanism that serves as a launcher for a few applications: the Quick Launch bar. This is the shortcut area that appears on your task bar, typically beside the Start button. If you don't see it, you'll have to turn it on by right-clicking on the taskbar and choosing the Quick Launch bar. You can drag-and-drop shortcuts here and use it as a launcher. And, because this is a directory (like everything else), you can place things directly in the Quick Launch folder. Just like all other shortcuts, you may assign operating system–wide key accelerators to these items, but existing application accelerators will interfere with them.

> NOTE
> **Typing is faster than navigation.**

Windows Vista has a slightly new twist to the Quick Launch bar. You can run the applications associated with the shortcut via the Windows-<NUM> keysym. In other words, Windows-1 selects and presses the first Quick Launch item, Windows-2 launches the second, and so on. This mechanism works great…as long as you have only 10 applications that you use regularly! While this won't accommodate as many applications as a real launcher, it might be a handy place to put a few critically important applications.

Copyrighted material

WHY NOT JUST ASSIGN HOTKEYS TO YOUR FAVORITE APPLICATIONS?

All the major operating systems allow you to create keyboard accelerators (that is, hotkeys) to launch applications. So, why not just define a list of accelerators and be done with all this launching business? Mapping hotkeys to application launching works great if you always have the desktop as the current focus. But developers virtually never have only the desktop (or filesystem explorer) open. Typically, a developer has 20 special-purpose tools open, each with its own magic keyboard combination. Trying to use the operating system's hotkeys to launch applications contributes to this Tower of Babel effect. It is virtually impossible to find hotkeys that don't interfere with at least some of your currently open applications. While using operating system–level hotkeys sounds like an attractive idea, it falls down in practical use.

Mac OS X

Mac OS X's *dock* combines the utility of the quick start menu and the task buttons in Windows. It encourages you to place oft-needed applications on the dock, and drag the others off into space (with a satisfying "poof" when they disappear). Just like with the quick start bar, the constraints of real estate hurt you: placing a useful number of applications on the dock expands it to the point where it becomes cumbersome. This has created a cottage industry of alternative launchers for Mac OS X. Even though some well-known launchers have been around for years, most power users today have migrated to Quicksilver.

Quicksilver[‖] is the closest anyone has come to creating a graphical command line. Just like a bash prompt, Quicksilver allows you to launch applications, perform file maintenance, and a host of other behaviors. Quicksilver itself appears as a floating window, invoked via a customizable hotkey (everything about Quicksilver is customizable, including the look and feel of the floating window). Once it appears, you can perform actions in the "target" pane.

GETTING QUICKSILVER

Quicksilver is currently free, downloadable from *http://quicksilver.blacktree.com/*. Quicksilver's creator is actively encouraging developers to build more plug-ins for it. There are plug-ins for Subversion, PathFinder, Automator, and tons of other applications, both core operating system and third-party.

Quicksilver is absolutely addictive. More than any other piece of software, it has fundamentally changed the way I interact with my computer. Quicksilver represents that rarest of commodities in

[‖] Download at *http://quicksilver.blacktree.com/*.

Copyrighted material

software: simple elegance. The first time you see it, you think "No big deal, just a new way to launch applications." The more you use it, however, the more subtlety you see, and the more its power gradually reveals itself.

Quicksilver makes it easy to launch applications via a hotkey and a couple of keystrokes. You get three panes in Quicksilver: the top one for files or applications ("nouns"), the middle for actions ("verbs"), and the third (if necessary) for the target of the action ("direct object"). When you search for items in Quicksilver, it treats everything you type as it if had a universal wildcard character. For example, if you type "shcmem" in Quicksilver, it will locate a file named *ShoppingCartMemento.java*.

Quicksilver doesn't just launch applications. It allows you to apply any (context-sensitive) command to any file. In Figure 2-3, I've selected a file named *acceleration_quicksilver_regex.tiff* and I've specified the Move To… action. The third pane allows me to choose the destination for the move, in the same manner that I chose the filename in the target pane (that is, using the special wildcard behavior described earlier).

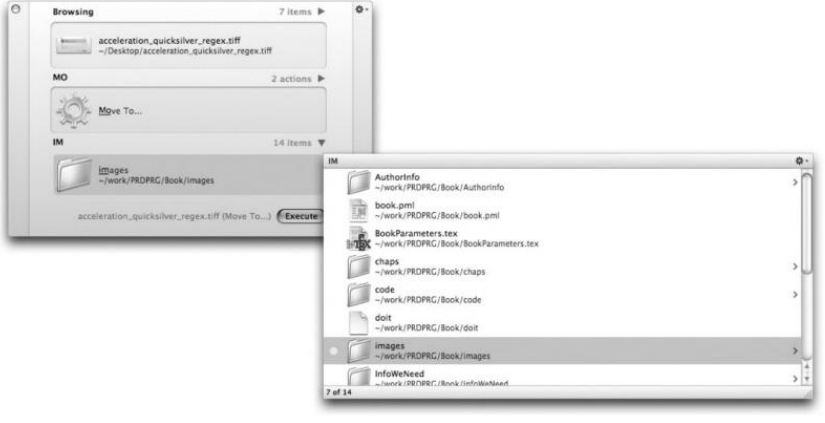


FIGURE 2-3. Quicksilver's third pane indicates a move target

Why is this such a big deal for developers? Quicksilver works via plug-ins, and a fair number of developer-centric plug-ins exist. For example, Quicksilver features excellent Subversion integration. You can update repositories, commit changes, get status, and a whole bunch of other functions. While not as powerful as command-line Subversion (nothing really compares to that), it gives you a quick graphical way to do common chores with just a few keystrokes.

One other thing about Quicksilver is worth mentioning: *triggers*. A trigger is a noun-verb-direct object combination, just as you would do it via the normal user interface, stored permanently

Copyrighted material

under a hotkey. For example, I have several projects that I use the same sequence of keys for all the time:

1. Invoke Quicksilver
2. Choose the directory of the project (the noun)
3. Choose the "Open with…" action (the verb)
4. Choose TextMate as the application (the direct object)

I do this so often I've assigned a trigger to it. Now, with a single keystroke (Alt-1 in my case), I can invoke this sequence of commands. Triggers are designed to allow you to save common Quicksilver operations under a single hotkey. I also use triggers for things like starting and stopping servlet engines (like Tomcat and Jetty). Very useful indeed.

I've only scratched the surface of Quicksilver's capabilities. You can launch applications, apply commands to one or multiple files, switch songs in iTunes, and tons of other stuff. It changes the way you use the operating system. With Quicksilver, you can use the dock just as a task manager, showing the currently running applications, and use Quicksilver as the launcher. Quicksilver customization happens through a published plug-in API (see the previous sidebar "Getting Quicksilver" for information about how to download Quicksilver and its plug-ins).

Quicksilver is a great example of an application that looks too simple to be useful when you first install it. Numerous friends have said to me "I installed Quicksilver, now what do I do?" To that end, some friends and I created a blog around general productivity topics on the Mac, called PragMactic-OSXer (*http://pragmactic-osxer.blogspot.com*).

WHY NOT JUST USE SPOTLIGHT?

The functionality of Quicksilver overlaps that of Spotlight, the built-in search facility on Mac OS X. But Quicksilver is much more than just quick search. It allows you to essentially replace the Mac OS X Finder, because all typical file manipulation is done more quickly in Quicksilver (typing is faster than navigating). Quicksilver allows you to specify what items you want to catalog (unlike Spotlight indexing your entire hard drive), making Quicksilver faster at finding files in its index. And Quicksilver uses the cool "regex between every character" way of specifying search items, which Spotlight doesn't. I virtually never use Finder anymore. All file operations (and pretty much all interaction with my computer) happen through Quicksilver. I've become so dependent on it that if it ever crashes (which happens rarely but does happen; it's still in beta, after all), it's as if my machine has been suddenly crippled. More than any other utility I've ever used, it has changed the way I work.

Leopard's version of Spotlight is much faster than previous versions, but of course these two tools aren't mutually exclusive. In Leopard's version of Spotlight, you can now do searches across multiple machines (which Quicksilver won't do). For this to work, you must be logged into the other machine (for obvious security reasons). Now, when you perform a Spotlight search, you can choose on the toolbar which machine you want to search. In the example shown in Figure 2-4, from my laptop I've logged onto the desktop machine (called *Neal-office*) and selected the home directory (named

Copyrighted material

nealford). When I do the Spotlight search, I can choose the target in the toolbar at the top. The file *music.rb* exists only on the desktop machine.

FIGURE 2-4. Spotlight on Leopard allows cross-machine searching

Unfortunately, the Windows and Linux worlds don't have anything quite as spectacular as Quicksilver. The aforementioned Colibri implements a small percentage of Quicksilver's functionality (mostly around its launching ability, but not the graphical command-line part). Hopefully, someone will eventually either port Quicksilver to other platforms or create a convincing clone of it. It is far and away the most sophisticated launcher of any operating system.

Launching in Linux

Most desktop Linuxes run either GNOME or KDE. Both have similar taskbar style user interfaces borrowed from Windows. However, it is considerably more difficult to customize their start options because their menu structures don't exist as simple directory entries. Modern versions of GNOME include a pretty functional launcher, tied by default to the Alt-F2 keysym. It shows a list of the runnable applications and allows you to refine your selection via typing. In Figure 2-5, the list is narrowed by the two letters "fi."

Accelerators

NOTE
Prefer typing over mousing.

Developers are essentially specialized data entry clerks. Instead of coming from some external source, the data we enter into the computer comes from our heads. But the lessons of data entry operators still resonate. Data entry workers who are paid for the amount of information

Copyrighted material

FIGURE 2-5. GNOME's "Run Application" launcher

they can input know that using the mouse slows them down by orders of magnitude. Developers can learn an important lesson here.

The classic "No Mouse Required" application is the VI editor. Watching an experienced VI user inspires awe. The cursor seems to follow their eyes. Unfortunately, it takes about two years of daily VI usage to get to that point because the learning curve is so daunting. If you've used it every day for 1 year and 364 days, you'll still struggle. The other classic Unix editor is Emacs, which is also very keyboard-centric. Emacs, though, is the proto-IDE: through its plug-in architecture, it does a lot more than just edit files. VI users scornfully refer to Emacs as "a great operating system with limited text-editing capabilities."

Both VI and Emacs support a very important accelerator: never taking your hands off the character keys. Even reaching down to the arrow keys on a keyboard slows you down because you must return to the home row keys to type characters again. Really useful editors keep your hands in optimum position to input and navigate at the same time.

Short of learning VI, you can figure out how to use *accelerators* to speed up your interaction with both the operating system and its applications. This section describes some ways to accelerate your use of both underlying operating systems and tools like IDEs. I start at the operating system–level and work my way up the food chain toward IDEs.

Operating System Accelerators

Graphical operating systems favor convenience (and eye candy) over raw efficiency. The command line is still the most efficient way to interact with a computer because very little

Copyrighted material

stands between the user and the desired result. Still, most modern operating systems support a plethora of keyboard shortcuts and other navigation aids to help speed up your work.

Windows address bar

> NOTE
> The address bar is the most efficient Windows Explorer interface.

One of the great navigation aids on command lines is autocompletion, where you hit the Tab key and the shell automatically completes the matching element in the current directory. If more than one match occurs, it generates the common parts and allows you to add more characters to complete a full name of something (a directory, a filename, etc.). All the major operating systems have completion on command lines now, usually using the Tab character.

WHAT IF I'M STILL RUNNING WINDOWS 2000?

Even though Windows 2000 doesn't perform tab filename completion at the command line by default, it is a simple registry tweak away. To enable filename completion in Windows 2000:

1. Run regedit.
2. Navigate to *HKEY_CURRENT_USER\Software\Microsoft\Command Processor*.
3. Make a DWORD value EnableExtensions, equal to 1.
4. Make a DWORD value CompletionChar, equal to 9.

Many developers don't realize that Windows Explorer's address bar also offers Tab filename completion, exactly like a command prompt. The keyboard shortcut to get to the address bar is Alt-D; from there, you can start typing part of a directory, hit Tab, and Explorer will complete the name for you.

Mac OS X Finder

Mac OS X includes a huge number of keyboard shortcuts, and each application has a set of its own. Ironically enough, considering Apple's general concern for usability, OS X applications aren't as consistent as those in most Windows applications. Microsoft has done a great job of creating and enforcing some common standards, and key mapping has probably been its biggest success. Nevertheless, Mac OS X has some nice built-in keyboard shortcuts and others that aren't immediately apparent. Like much around the Mac, it takes someone showing you many of these before you discover them.

A perfect example of this is keyboard navigation in both Finder and in open/save dialogs. In Windows Explorer, the address bar is obvious. In Finder, though, you can use tab completion

Copyrighted material

to navigate to any folder (just like using the address bar in Explorer) by typing Apple-Shift-G, which displays a dialog into which you can type the location.

You shouldn't use either Finder or the terminal exclusively (see "Command Prompts at Your Fingertips," later in this chapter). They interact quite nicely with one another. You can drag folders from Finder into Terminal for a quick way to issue a *cd* command. You can also use the *open* command to open files from Terminal rather than double-clicking them in Finder. It's all about context and learning the capabilities of the tools at hand so that you can apply those capabilities appropriately.

> NOTE
> **Take the time to learn all the hidden keyboard shortcuts of your universe.**

Shortcuts that Windows users miss a lot on Mac OS X are the Alt key accelerators for applications. Mac OS has them, but they're based on incremental search rather than explicit key relationships. The Ctrl-F2 key moves focus up to the menu bar, and you can type the first part of the menu item you want. When it's highlighted, hit Enter and start incrementally typing the enclosed menu item. It sounds complicated, but it works beautifully, and it works across all applications. You can also use Ctrl-F8 to move the focus to the far right of the menu bar, where all the services icons live.

My biggest problem with this was the cumbersome gymnastics needed to invoke Ctrl-F2, so I used the standard Mac OS X keyboard shortcut dialog to remap it to Ctrl-Alt-Apple-Spacebar (which sounds even worse, but they all line up, so it's an easy combination to hit). Plus, my Quicksilver invoker is mapped to Apple-Enter, so all my "meta" navigation is mapped to more or less the same general area.

If you are using the latest version of Mac OS X, choosing menu items is even easier. One of Leopard's help features finds menu items for you when you type the name (or just part of the name). This is a great way to access menu items that live in deeply nested menus, ones whose names you remember but whose locations evade you, and things that you think the application should do but you don't know where the functionality lives. If you hit the Apple-? key, the help search option will appear. Type any part of the menu item name you want, and Leopard will highlight it for you and invoke it if you hit Enter. As is the case with much keyboard magic, it is harder to explain than to do (Figure 2-6).

Clipboard(s)

It is sometimes amazing how far backward we can fall. Both of the legendary editors of yesteryear (VI and Emacs) have multiple clipboards (also called registers). Yet, the two major operating systems limit us to a measly single clipboard. You'd think that clipboards were a scarce natural resource that must be doled out carefully lest we run out someday. This is a perfect example of how much useful information we lose from generation to generation of

Copyrighted material

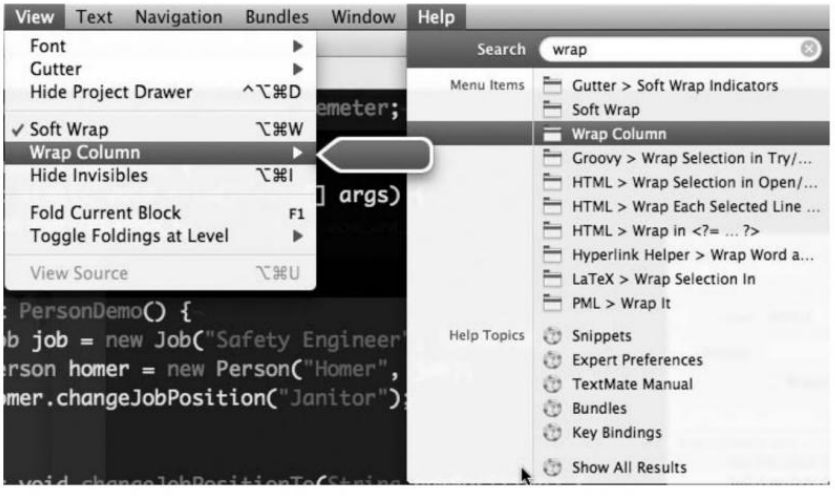


FIGURE 2-6. Leopard finds menu items for you

developers because of insufficient lore between groups. We keep reinventing the same things over and over because we don't realize that someone already solved the problem a decade ago.

> NOTE
> **Context switching eats time.**

Having multiple clipboards may not seem like a big productivity gain. But once you get accustomed to having them, it changes the way you work. For example, if a task requires copying and pasting several disjointed items from one file to another, most developers will copy, bounce to the other file, paste, bounce back to the first one, and repeat ad nauseum. Clearly, this isn't a productive way to work. You end up spending too much time context switching between the open applications. If you have a clipboard *stack*, however, you can harvest all the values from the first file—stacking them up on your clipboards—then bounce once to the destination file and paste them all in the appropriate places, one at a time.

> NOTE
> **Clipboarding in batches is faster than clipboarding serially.**

It's interesting how such a simple mechanism takes time to internalize. Even when you install a multi-clipboard utility, it takes a while before you realize all the situations where it applies. Too often, you install it and promptly forget it's there. Like with many of the productivity hints in this book, you have to keep an active mindset to take advantage of these techniques. Recognizing the appropriate situation where one of them applies is half the battle. I constantly use clipboard history; I can't imagine life without it now.

Copyrighted material

Fortunately, there are a variety of clipboard enhancers available for both Windows and Mac OS X, both open source and commercial. A nice, bare-bones open source alternative in Windows is CLCL,[#] which gives you a configurable clipboard stack and allows you to assign your own keyboard shortcuts. For Mac OS X, JumpCut[*] is a simple, open source clipboard stack. For a more elaborate (commercial) offering, jClip[†] is very nice, allowing you not only a clipboard stack but also a configurable number of clipboards, distinct from one another. Having unique clipboards is nice if you have a large group of items you are copying and you don't want to pollute your main clipboard stack.

Be careful when you get accustomed to using a clipboard stack, though. You might exuberantly start talking about it to a be-sandaled Unix guy, causing an hour-long lecture about how he's had multiple clipboards since you were in grammar school and all the other ways that 20-year-old text editors rule.

Remember History

> NOTE
> **Those who remember their history aren't doomed to type it again.**

All shells have a history mechanism, which allows you to recall previous commands and repeat them, with changes if necessary. This is one of the huge advantages that shells have over graphical environments: you can't repeat actions with subtle changes easily in graphical environments. Thus, learning to perform operations at the command prompt means that you have a more effective communication layer between yourself and the machine.

History is generally wired to the up and down arrow keys, which is a brute-force way to get to previous commands. But, as I've stated before, searching is more effective than navigation. You can search your history to find the command in question faster than you can scan each entry as you walk back through them one at a time.

In Windows, type the first part of the previous command and then hit F8. The shell will perform a backward search through previous commands that match the first part of what you've just typed. You can keep hitting the F8 key to continue walking up the list of matching commands. If you want to see the command history, type F7, which shows your recent history in a list where you can utilize the up and down arrow keys to select the command.

On Unix-based systems (including Cygwin), you can choose which type of command-line key syntax you want, either Emacs (usually the default) or VI. As I mentioned previously, VI is a

[#] Download at *http://www.nakka.com/soft/clcl/index_eng.html.*

[*] Download at *http://jumpcut.sourceforge.net/.*

[†] Download at *http://inventive.us/iClip/.*

Copyrighted material

super powerful navigation keyset, but it is very daunting to learn from scratch. You can set VI mode in your *-nix environment by adding the following to your ~/.profile file:

```
set -o vi
```

When you have VI mode set, you can hit Escape (which puts you in command mode), then / to put yourself in search mode. Type the search text, then hit Enter. The first match will be the most recent command that matches the search string. If that's not the one you want, hit / followed by Enter to search for the next occurrence. Also in bash, if you have recently executed a command, you can hit the hotkey ! along with the first letter of the recent command to re-run it. The ! gives you access to the history directly. If you want to see your command-line history, execute the *history* command, which provides a numbered list of the commands you've executed, in reverse order (in other words, the most recent command is at the bottom of the list). You can execute commands from the history using an exclamation point (!) + the history number of the command you want. This is great if you have some complex command you want to re-execute.

There and Back

As developers, we constantly jump around the filesystem. We always need to grab a JAR file, go find some documentation, copy an assembly, or install a foo over at the bar. Consequently, we must hone our navigation and targeting skills. As I've preached for a while in this chapter, graphical explorers and finders are poorly suited to that kind of jumping around (because it's never just a jump…it's a round trip somewhere to handle some little chore, because we'll have to go back to where we started).

HIDDEN ALT-TAB ENTRIES

The Alt-Tab viewer in Windows holds only 21 items. Once you've exceeded that number, they just stop showing up (even though the applications are still running). You can either control your Explorer spawning or follow one of these two solutions, both involving Windows PowerToys. The first is Tweak UI, which allows you to configure the number of items that appear in the Alt-Tab dialog. The other solution involves installing multiple desktops through the Virtual Desktop PowerToy, which I discuss in "Segregate Your Workspace with Virtual Desktops," in Chapter 3.

Mac OS X allows you to kill instances of applications while you are Apple-Tab'ing around… just hit the Q key with the doomed application focused and it will close. Similarly, if you use the application manager Witch, you can kill individual windows using the W key while the window is focused, which is great for killing Finder windows left and right. Of course, if you use Quicksilver, you don't need Finder windows as much.

Copyrighted material

A couple of old-school command-line tools offer a good alternative to spawning new Explorers every time you need to jump to another location. They allow you to navigate temporarily to another location, do whatever needs to be done, and then return to where you started. *pushd* performs two actions: it puts you in the directory you've passed as an argument and pushes the current directory into an internal stack. *pushd* is therefore an alternative to the more pedestrian *cd* command. Once you've finished your work, issue a *popd* command to return to the original location.

pushd and *popd* work on a directory stack. This is a stack in the computer science sense, meaning that it acts as a FILO (First In, Last Out) list (the classic, well-worn analogy is a stack of dishes in a cafeteria). Because it is a stack, you can "push" as many times as you want, and they will "pop" back off in the reverse order.

All Unixes have the *pushd* and *popd* commands (including Mac OS X). However, they aren't some esoteric Cygwin-only adjunct to Windows.

```
[jNf| ~/work ]=> pushd ~/Documents/
~/Documents ~/work
[jNf| ~/Documents ]=> pushd /opt/local/lib/ruby/1.8
/opt/local/lib/ruby/1.8 ~/Documents ~/work
[jNf| /opt/local/lib/ruby/1.8 ]=>
```

In this example, I started in the ~/work directory, jumped over to the ~/Documents directory, and was then off to the Ruby install directory. Each time the *pushd* command is issued, it shows the existing directories already on the stack. This is true on all the Unix flavors I've tried. The Windows version of this command performs only the first two of the tasks above: it doesn't give you any clue of what's currently on the stack. However, this isn't a huge burden when using it, since this pair of commands is mostly used to make a quick jump to another place, then right back.

Command Prompts at Your Fingertips

Imagine yourself on the Productive Programmer therapy couch. "I'd really *like* to spend more time at the command line, but most of the stuff I need to do is in Explorer." Well, I'm here to help. A couple of ways exist to make it quick and easy to move back and forth between the graphical view and the command prompt.

Command Prompt Explorer Bar

NOTE
Embedded command prompts give you access to the best of both worlds.

Copyrighted material

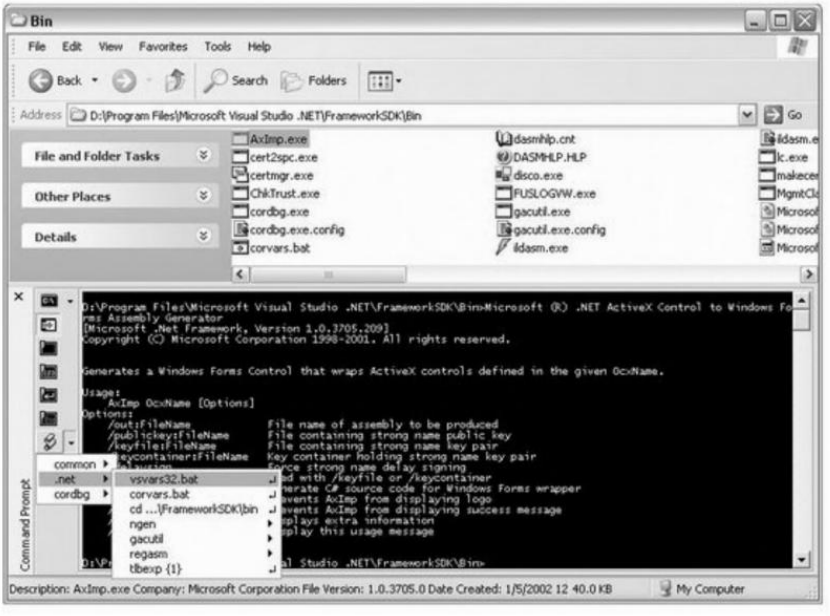


FIGURE 2-7. Command Prompt Explorer Bar

For Windows, Command Prompt Explorer Bar‡ is a great open source utility that allows you to open a command prompt attached to the bottom of your current Explorer view, using the keyboard shortcut Ctrl-M. See this utility in action in Figure 2-7. One of the great usability features of this tool is its "stickiness" for the directory shown in the attached Explorer view. When you change directories in Explorer, the directory automatically changes in the command prompt below. Unfortunately, the relationship isn't two-way: changing the directory in the command prompt window does not change it in the Explorer view. Nevertheless, it is a useful utility.

Unfortunately, Mac OS X doesn't have any native ability to perform this trick. However, the commercial Finder replacement Path Finder§ can make it happen, as shown in Figure 2-8. This terminal is like any other terminal window in Mac OS X (it reads your home profile, etc.), and it is launchable with the keyboard shortcut Alt-Apple-B. Once you become accustomed to having a terminal (or command prompt) so easily accessible, you tend to use it more for appropriate tasks.

The graphical view of the directory structure (Explorer, Finder) can also interact with the command-line view (command prompt, terminal) in a related but obscure way via drag-and-

‡ Download at *http://www.codeproject.com/csharp/CommandBar.asp.*

§ Download at *http://www.cocoatech.com/.*

Copyrighted material

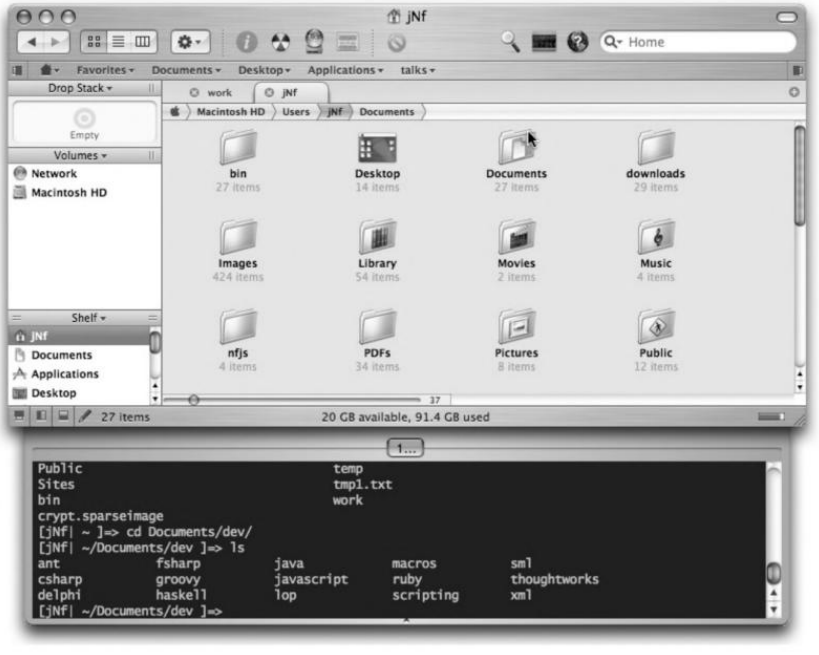


FIGURE 2-8. Path Finder's attached terminal

drop. In both Windows and Mac OS X, you can drag a directory into a command-line view to copy the path. Thus, if you want to change to a certain directory in the command prompt and you have Windows Explorer open to the parent folder (or anywhere that you can grab the target directory), type *cd* and then drag the directory to the command prompt and it will fill in the name of the directory. You can also use *Command Prompt Here*, discussed in the next section.

And one more cool trick in Mac OS X. If you go to Finder and copy files, you can access them in a terminal window by performing a paste operation (Apple-V); Mac OS X brings the whole path with the filename. You can also interact with the clipboard for piping operations using *pbcopy* (which copies stuff to the clipboard) and *pbpaste* (which pastes stuff from the clipboard to the command line). However, note that *pbpaste* just pastes the filename, not the entire path.

Here!

NOTE
Embed the command prompt with Explorer to make it easier to switch contexts.

Copyrighted material

I have one last entry in this family of acceleration tools. If you take the time and trouble to navigate the long, torturous route to get to a directory in Windows Explorer, you don't want to walk the same path again in a command prompt. Fortunately, one of the Microsoft PowerToys can come to your rescue: *Command Prompt Here*. Installing this PowerToy makes a few tweaks to the registry and adds a context (that is, right-click) menu called, as you can probably guess, "Command Prompt Here." Executing this command opens a command prompt in the directory you have selected.

POWERTOYS

Microsoft issues (but doesn't support) a collection of utilities known as PowerToys.[‖] These power toys add all sorts of interesting capabilities to Windows, going all the way back to Windows 95, and they are all free. A lot of them (like Tweak UI) are really just dialogs that do special tweaks to registry entries. Some PowerToys include:

Tweak UI
: Allows you to control all sorts of visual aspects of Windows, like what icons appear on the desktop, mouse behaviors, and other hidden capabilities.

TaskSwitch
: Improved task switcher (tied to Alt-Tab, which shows thumbnails of the running applications).

Virtual Desktop Manager
: Virtual desktops for Windows (see "Segregate Your Workspace with Virtual Desktops" in Chapter 3).

Microsoft doesn't support these little utilities. If you've never played with them, go look at the list. Chances are pretty good that something you've always wanted Windows to do for you is already there.

Not to be outdone, you can obtain a *"Bash Here"* context menu by running *chere* from Cygwin, which opens a Cygwin bash shell at that location instead of a command prompt. These two tools play together nicely, so you can install both and decide on a case-by-case basis whether you want a command prompt or a bash shell. The command:

```
chere -i
```

installs the "Bash Here" context menu and:

```
chere -u -s bash
```

uninstalls it. Actually, the *chere* utility will also install a Windows command prompt "Command Prompt Here" context menu (just like the Windows PowerToy) via the command:

‖ Download at *http://www.microsoft.com/windowsxp/downloads/powertoys/xppowertoys.mspx*.

Copyrighted material

```
chere -i -s cmd
```

So, if you have Cygwin, there is no need to download the "Command Prompt Here" PowerToy, just use *chere*.

Path Finder in Mac OS X also has an "Open in Terminal" context menu option, which opens yet another terminal window (not the drawer version depicted in Figure 2-8 but a full-blown separate window). And Quicksilver has an action called "Go to the directory in Terminal."

Development Accelerators

Pop quiz: what's the biggest clickable target on your screen? It's the one right under your cursor, which is why the right-mouse menu should have the most important things on it. The target right under your mouse is effectively infinitely large. Second question: what's the next biggest target? The edges of the screen because you can accelerate as fast as possible to the edge and not overshoot it. This suggests that the really important stuff should reside on the edges of the screen. These observations come from Fitt's Law, which states that the ease of clicking on a target with the mouse is a combination of the distance you must navigate and the size of the target.

The designers of Mac OS X knew this law, which is why the menu bar resides at the top of the screen. When you use the mouse to click one of the menu items, you can ram the mouse pointer up against the top of the screen and you are where you want to be. Windows, on the other hand, has a title bar at the top of each window. Even if the window is maximized, you still must carefully find your target by accelerating to the top, and then use some precision mousing to hit the target.

There is a way to mitigate this for some Windows applications. The Microsoft Office suite has a "Full Screen" mode, which gets rid of the title bar and puts the menu right at the top, like Mac OS X. There is help for developers, too. Visual Studio features the same full-screen mode, as does IntelliJ for Java developers. If you are going to use the mouse, using your applications in full-screen mode makes it easier to hit the menus.

But speeding up the use of the mouse isn't really what I advocate. Programming (except for user interface design) is a text-based activity, so you should strive to keep your hands on the keyboard as much as possible.

> NOTE
> **When coding, always prefer the keyboard to the mouse.**

You use an IDE all day long to create code, and IDEs have a huge number of keyboard shortcuts. Learn them all! Using the keyboard to get around in source code is always faster than using a mouse. But the sheer number of keyboard shortcuts can be intimidating. The best way to learn them is to make a conscious effort to internalize them. Reading big lists isn't helpful because the shortcuts aren't in context. The Eclipse IDE has a nice shortcut key that shows all the other

Copyrighted material

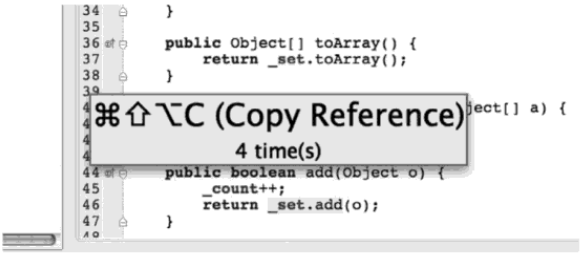


FIGURE 2-9. Key Promoter is IntelliJ's useful nag about shortcuts

shortcut keys for a particular view: Ctrl-Shift-L. This is a great mnemonic because it's already in the appropriate context. The best time to learn keyboard shortcuts is when you need to perform the activity. When you go to the menu, notice the keyboard shortcut on it. Then, instead of selecting the menu item, remember the shortcut, dismiss the menu, and do it on the keyboard. That will reinforce the association between the task and the keyboard shortcut. And believe it or not, saying the shortcut aloud also helps because it forces it into more parts of your brain. Of course, your coworkers might think you're insane, but you'll be able to out-keyboard them in no time.

One of my colleagues has a great way of teaching keyboard shortcuts. Whenever you pair-program with him, if you use a mouse to select a menu or tool button, he makes you undo your operation, then do it three times using the keyboard. Yes, it slows you down at first, but the reinforcement (plus his evil eye when you forget them) is a pressure cooker to learn shortcuts.

> NOTE
> **Learn IDE keyboard shortcuts in context, not by reading long lists.**

Another great way to remember shortcuts is to have someone (or something) pester you about them. IntelliJ has an awesome plug-in called Key Promoter. Every time you use the menu to select something, a dialog pops up that tells you the shortcut you could have used and how many times you've done it wrong (see Figure 2-9). The same utility, called Key Prompter, exists for Eclipse as well.[#] Key Prompter goes even further: you can set a mode that will *ignore* menu selections, forcing you to use the shortcut key!

Unfortunately, lots of great shortcuts don't live on menu items at all: they are buried in the massively long list of possible keyboard shortcuts. You should ferret out the cool ones for the IDE you use. Table 2-1 is a short list for Java developers of some cool, hidden keyboard shortcuts in both IntelliJ and Eclipse for Windows.

[#] Download at *http://www.mousefeed.com.*

Copyrighted material

TABLE 2-1. Choice keyboard shortcuts for IntelliJ & Eclipse

Description	IntelliJ	Eclipse
Goto class	Ctrl-N	Ctrl-Shift-T
Symbol list	Alt-Ctrl-Shift-N	Ctrl-O
Incremental search	Alt-F3	Ctrl-J
Recently edited files/files open	Ctrl-E	Ctrl-E
Introduce variable	Ctrl-Alt-V	Alt-Shift-L
Escalating selection	Ctrl-W	Alt-Shift-Up Arrow

A couple of these entries require a little further explanation. The *recently edited files/files open* entry works differently across the two IDEs: in IntelliJ, it provides a list of files you've edited recently, in the reverse order of access (so that the most recent file is at the top). In Eclipse, the keyboard shortcut provides a list of open buffers. This is important to developers because we tend to work in a small clustering of files on a regular basis, so ready access to a small group helps.

Introduce variable is technically a refactoring function, but I use it constantly to type the lefthand side of expressions for me. In both IDEs, you can type the righthand side of an expression (such as `Calender.getInstance();`) and let the IDE supply the lefthand side (in this case, `Calendar calendar = `). The IDE can supply variable names almost as well as you can, and it's much less typing and thinking about what to name variables. (This shortcut has made me particularly lazy when coding in Java.)

The last special entry is *escalating selection*. Here's how it works. When you place your cursor on something and invoke this command, it extends the selection one level toward the next higher syntactic element. The next time you hit they key, it broadens the selection to the next larger group of syntactic elements. Because the IDE understands Java syntax, it knows what constitutes a token, a code block, a method, etc. Instead of creating a half-dozen shortcuts to select each of those elements, you can use the same keystroke over and over to gradually widen the selection. It's cumbersome to describe, but try it, you'll quickly fall in love.

Here is the way to learn *and internalize* the really cool keyboard shortcuts you saw in the Giant Long List of keyboard shortcuts. Read the list one more time, but copy the really useful ones you didn't know about to a separate file (or even paper!). Try to remember that the capability exists, and the next time you need it, look at your cheat sheet. This represents the missing link between "I know that it can be done" to "this is how to do it."

The other key to IDE productivity is *live templates*. These are snippets of code that represent some chunk of code you use all the time. Most IDEs allow you to parameterize your templates, filling in values when the template is expanded in your editor. For example, here is a parameterized template in IntelliJ that allows you to iterate over an array in Java:

```
for(int $INDEX$ = 0; $INDEX$ < $ARRAY$.length; $INDEX$++) {
  $ELEMENT_TYPE$ $VAR$ = $ARRAY$[$INDEX$];
```


Copyrighted material

```
    $END$
}
```

When this template is expanded, the IDE first places the cursor on the first `$` delimited value, allowing you to type in your index name, then a tab takes you to the next parameter. In this template language, the `$END$` marker is where the cursor will end after all the expansions.

Every IDE has a different syntax for this, but virtually every IDE worth using supports this concept. Learn the template language in your IDE and use it as much as you can. Outstanding template support is one of the reasons for the popularity of the TextMate * and E-Text† editors. The template doesn't make typos, and having templates around for complex language constructs saves you time and mental energy while you code.

> NOTE
> **When you type a complicated construct for the second time, templatize it.**

Search Trumps Navigation in Tools, Too

Code hierarchies have also gotten too deep to be useful. Once they reach a certain size, filesystems, package structures, and other hierarchical systems become too deep for effective navigation. Large Java projects suffer from this because the package structure is tied to the directory structure. Even for a small project, you must dig through trees—expanding nodes as you go—to find a file, even if you already know the name of it. If you find yourself doing this, you are working too hard for your computer.

Modern Java IDEs allow you to quickly find any Java source files within the current project by typing Ctrl-N on Windows or Apple-N on the Mac (for IntelliJ) and Ctrl-Shift-T (for Eclipse). The example shown in Figure 2-10 is from IntelliJ; it opens a text box in the editor, allowing you to type the name of the file you want.

FIGURE 2-10. IntelliJ's "find files" text box

Typing in the entire name (or even a significant portion of it) is cumbersome. It would be nice if the IDE were even smarter about how you specify names. And it is. Instead of typing the name of the file, if you start typing capital letters, it looks for names that have that same pattern of capital letters. For example, if you are looking for the file *ShoppingCartMemento*, you can

* *Download at http://macromates.com/.*

† *Download at http://www.e-texteditor.com/.*

Copyrighted material

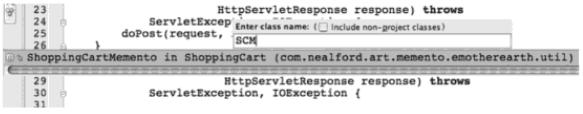


FIGURE 2-11. IntelliJ's smart pattern matching for names

type SCM and the IDE will ignore the intervening lowercase letters and find the matching pattern of capital letters, as shown in Figure 2-11.

This file-finding magic works with non-Java source files, too (add a Shift to the other keys in IntelliJ, or use Ctrl-Shift-R in Eclipse). This is the "find resources" textfield, and it works just like the "find files" one. Don't go slogging through the huge tree of source files anymore: you know what you want, so go directly to it.

For .NET developers, the common environment is Visual Studio 2005 (in its current incarnation). While it has a middling number of keyboard shortcuts, you can super-charge it with the commercial Resharper (from JetBrains, the creators of the IntelliJ Java IDE). Many developers think that Resharper concerns itself primarily with adding refactoring support, but savvy developers realize that it also adds a huge number of keyboard shortcuts (including the "find files" capability described earlier).

Macros

Macros are recorded snippets of interaction with your computer. Generally, each tool comes with its own macro recorder (because only the tool knows how it processes keys). That means, of course, that there is no standard macro syntax, sometimes even between different versions of a product. For years, Microsoft Word and Excel had very different macro syntax even though they were from the same company and in the same Office suite. It wasn't until Office 2000 that Microsoft finally agreed on a single syntax. Even though a Tower of Babel exists between tools, macros can still help solve very specific problems that you face on a daily basis.

Macro Recorder

> NOTE
> **For any symmetric operation on multiple lines of text, find a pattern and record a macro.**

How often do you find yourself working in a pattern? You've cut and pasted some information from an XML document, and now you have to remove all the XML cruft from around the real data inside to clean it up. Macros used to be all the rage with developers, but they seem to have fallen out of favor recently. I suspect that the live template feature of most modern IDEs has removed some of the need for macros.

Copyrighted material

But, however much you lean on live templates, uses still exist for recording macros. The common scenario is the one highlighted earlier: doing one-time processing on some information to either de-cruft it from another format or cruft it up for the consumption of some other tool. If you can shift your perspective on the task and see it as a series of repeatable steps, you'll find macros can be used for lots of chores.

> NOTE
> **The more times you perform a particular operation on a chunk of text, the greater the likelihood you'll do it again.**

Even if you use Eclipse (which doesn't have a macro recorder), you can always jump over to a text editor and use its macro recorder for just this chore. One of the important selection criteria for a text editor is its macro recording facilities and the format of the recorded macros. It is nicer if the macros produce some kind of readable code that you can tweak by hand, creating a reusable asset you can use later down the road. After all, if you've cut and pasted some stuff from one format to another once, chances are good you'll have to do it again.

Key Macro Tools

While formal macros in editors are great for processing text, code, and transformations, another category of macro tools helps you out on a daily basis. All the major operating systems have open source and/or commercial *key macro* tools. Key macro tools run in the background, waiting for a text pattern to expand. They allow you to type an abbreviation instead of the full text of something. Mostly, these tools do things like automatically type salutations for email addresses. But, as developers, we type lots of repetitive text in places where we don't have live templates (like the command line or in web browsers).

> NOTE
> **Don't type the same commands over and over again.**

One of the tasks I have to perform all the time is showing people how to use Selenium's remote control feature. To make it work, you must start up a proxy server and issue cryptic commands to give it instructions, which are basically just incantations at the command line. I'm not in an IDE, so I can't use live templates or even macros. I can't even use batch or shell scripts: I'm running against an interactive proxy. It didn't take me long to realize that I should save these commands in my key macro tool:

```
cmd=getNewBrowserSession&1=*firefox&2=8080
cmd=open&1=/art_emotherearth_memento/welcome&sessionId=
```

The ugly line of code is issued after I start the proxy server for Selenium, in the very particular format that Remote Control Selenium requires. If you don't know anything about Selenium, these commands won't make sense. But making sense out of the commands isn't the point of the example. This is just one of the hideous command strings that I must type from time to

Copyrighted material

time. Every developer ends up with these, and they never make sense out of context (and frequently just barely when in context). But now, instead of copying and pasting this from somewhere, I just type *rcsl1* to generate the first line, *rcsl2* to generate the second, and so on for the 10 commands I need to show people.

Some key macro tools allow you to record keystrokes at the operating system–level and play them back (sometimes even capturing mouse clicks and other interactions). Others require you to type in the commands that you want associated with a particular macro. In both cases, you are capturing some operating system-level interaction that you have to do repeatedly in a format that makes it easy to reuse.

Key macro tools are also great for common phrases that you must type over and over. What about the text you have to type in Word for project status messages? Or entering hours in your time and expenses system? A key macro tool falls into the category of tools that you don't even know exists one day into the "how did I live without this?" category the next.

The most popular key macro tool for Windows is AutoHotKey[‡] (which is open source). Mac OS X has a couple in the "commercial but inexpensive category," like TextExpander[§] and Typinator.[‖]

Summary

It's one thing to talk about accelerating your computer interactions with launchers, clipboard managers, IDE shortcuts, and all the various things mentioned in this chapter. It's another to *apply* them. You know what speeds up your work, but you feel like you don't have time to apply it. "I know there's a keyboard shortcut that does this, but I'm in a hurry, so I'll use the mouse instead and look up that keyboard shortcut later." Later never comes. Becoming more productive involves finding a balance between pursuing productivity without gutting your current productivity (I know, this drips with irony). Try to tackle one productivity enhancer per week, concentrate on just that one until it becomes ingrained, then move on to the next. This method will have very little impact on your time, and you'll gradually accrue better productivity.

> NOTE
> **Spend a little time each day to make every day more productive.**

Applying acceleration has two contexts: knowledge of accelerators and the proper context in which to use them. For example, install a clipboard utility on your machine and force yourself

[‡] Download at *http://www.autohotkey.com/.*

[§] Download at *http://www.smileonmymac.com/textexpander/index.html.*

[‖] Download at *http://www.ergonis.com/products/typinator/.*

Copyrighted material

to think about it every time you need to copy and paste something. You will gradually start seeing situations where it saves you time because you are harvesting a set of copied values all at once, then pasting them as a group as well. Once you've internalized that utility, move on to another one. It's all about finding the balance of spending time to learn to become more productive.

Copyrighted material

CHAPTER THREE

Focus

Copyrighted material

THIS CHAPTER INTRODUCES A VARIETY OF WAYS TO ENHANCE YOUR FOCUS BY KILLING OFF inefficiencies and needless distractions. You probably suffer from lots of distractions at work, both from the computer itself and from the outside world. Here you will learn how to enhance your focus with specific tools and approaches to interacting with your computer, as well as ways to make your coworkers leave you alone so that you can quit banging rocks together and get some work done. The goal is to get you back to that dazed but happy state of just having scaled a virtual mountain.

Kill Distractions

You are a knowledge worker, meaning you are paid for the creative and innovative ideas you produce. Dealing with constant distractions, both at your desk and on your desktop, can threaten your best contributions to your projects. Developers crave a state known as *flow*, discussed in lots of places (it even has an entire book devoted to it, written by Csikszentmihalyi). All developers know this state: it's when you are so focused that time disappears, you develop an almost symbiotic relationship with the machine and the problem you are attacking. This is the state you've been in when you say, "Wow, have four hours passed? I didn't even notice." The problem with flow is that it is fragile. One distraction pulls you out, and it takes effort to get back in. It also suffers from inertia. Late in the day, you have to fight harder to get back to that state, and the more times you are abruptly pulled out, the harder it becomes. Distractions kill your focus on the problem at hand, making you less productive. Fortunately, you can effectively block distractions in a few simple ways.

> NOTE
> **The higher the level of concentration, the denser the ideas.**

Blocking Strategies

Concentration is hard to maintain, especially when your computer seems determined to drag your attention away from your work. Blocking visual and audible distractions helps you maintain a good, focused flow state. For audible distractions (especially if you don't have an office with a door you can close), you can wear headphones (even if you aren't listening to music). When others see you wearing headphones, they are less likely to interrupt you. If your office won't allow headphones, consider putting a "Do Not Disturb" sign on the entryway to your cubicle. That should make people think twice before barging in.

For visual distractions, turn off all the things on your machine that break your concentration. Email notification is very damaging because it creates artificial urgency. How many of the emails you receive in the course of a day really require an immediate response? Turn off your email client and check mail in batches, when you reach a natural breaking point in your work. This allows you to determine when you want to break your train of thought.

Copyrighted material

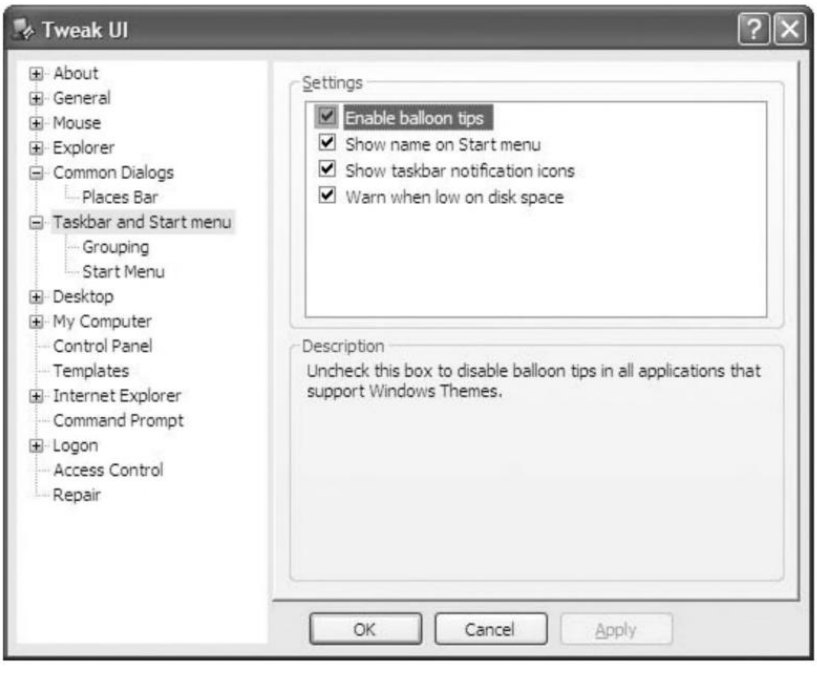


FIGURE 3-1. Killing balloon tips with Tweak UI

Turn Off Needless Notifications

Balloon tips in Windows and Growl notifications on Mac OS X also drag your attention away. On Mac OS X, Growl is customizable, so you can turn on only the notifications you need to see to get work done. Unfortunately, balloon tips in Windows are all-or-nothing. And lots of the messages balloon tips deliver are not useful. Do you really want to stop working to clean up the unused icons on your desktop? You also get messages about Windows automatically resizing virtual memory. I don't want to know that, and I especially don't want to be interrupted from my work to see it. Windows sometimes seems like a spoiled three-year old, always clamoring for attention.

There are two ways to turn off balloon tips. If you already have the Tweak UI PowerToy, one of the settings disables balloon tips, as shown in Figure 3-1. The other way involves a little registry editing (which is all the PowerToy is doing behind the scenes):

1. Run regedit.
2. Look for *HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced*.
3. Create a DWORD value (or edit it if it already exists) named EnableBalloonTips with value 0.

Copyrighted material

4. Log off and log in again.

If you tend to create lots of overlapping windows when you work, these can become distracting as well. There are a couple of freeware utilities available that are designed to "black out" the background, making all the applications you are not using fade out. This keeps your focus tightly bound to the task at hand.

For Windows, the application JediConcentrate* does the job. For Mac OS X, the fadeout application is called Doodim.† They both work the same way, allowing you to customize how much you want the background dimmed.

Create Quiet Time

If you work in an office with lots of other developers, consider instituting "quiet time," for example, from 9 A.M. to 11 A.M. and 3 P.M. to 5 P.M. During this time, everyone has their email turned off, there are no meetings, and it is verboten to call or go talk to someone unless there is an emergency (like you are blocked and can't get work done on the problem you are trying to solve). I tried this in one of the consulting offices where I worked, and it had an amazing outcome. Everyone in the office found that we got more done in those four hours than we were getting done in an entire day before we implemented the policy. All the developers started looking forward to it; it was everyone's favorite time of the day.

Another development team I know periodically books a meeting in their shared calendar. The "meeting" is really just time to get stuff done. Other people at the company can see from the shared calendar that everyone is "meeting" and therefore know not to disturb them. It is tragic that office environments so hamper productivity that employees have to game the system to get work done. Sometimes you have to think outside the box (or cubicle) to get work done despite your environment.

Search Trumps Navigation

> NOTE
> **The bigger the haystack, the harder it is to find the needle.**

Projects have gotten bigger and bigger, along with the packages and namespaces that go along with them. Hierarchies are hard to navigate when they get big: they're too deep. Filesystems that work well with 200 MB of storage suffer when they reach 200 GB. Filesystems have become massive haystacks, and we constantly do hard target searches for needles. Taking time out to dig around for files pulls you away from the problem upon which you should be focusing.

* Download at *http://www.gyrolabs.com/2006/09/25/jediconcentrate-mod/.*

† Download at *http://www.lachoseinteractive.net/en/products/doodim/.*

Copyrighted material

Fortunately, new search tools help you dispense with cumbersome filesystem navigation almost entirely.

Recently, powerful search applications appeared at the operating system–level: Spotlight in Mac OS X and Windows Search in Vista. These searching applications are different from the quaint search features in previous versions of Windows (whose only real use was to show an animation of a dog). This new breed of search tools indexes the interesting parts of your entire hard drive, making searches blazingly fast. They don't just look at filenames: they index the contents of files.

Several desktop search add-ons exist for pre-Vista Windows. My current favorite is the free Google Desktop Search.[‡] Out of the box, it searches only "normal" files (like spreadsheets, Word documents, email, etc.). One of the best parts of Google Desktop Search is its plug-in API, which allows developers to add search plug-ins. For example, Larry's Any Text File Indexer[§] allows you to configure Google Desktop Search for source files.

Once you've installed Larry's Any Text File Indexer and allowed it to index your hard drive (which it does in the background, during idle times), you can search for a snippet of the *contents* of a file. For example, in Java, the name of the file must match the name of the public class. In most other languages (like C#, Ruby, Python), the filename conventionally matches the class name. Or, if you are looking for all the files that use a particular class, you can search for code fragments you know exist. For example:

```
new OrderDb();
```

finds all the classes that create an instance of your `OrderDb` class.

Searching by contents is extremely powerful. Even if you can't remember the exact name of a file, you can almost always remember at least some of the content.

> NOTE
> **Replace file hierarchies with search.**

Indexing search utilities free you from the tyranny of the filesystem. Using a search tool like Google Desktop Search takes acclimatization because you likely have well-worn habits of searching for files by hand. You don't need this level of search for retrieving source files (your IDE already handles that for you). However, you constantly need to access a file where it lives, to perform some operation on it like version control, a diff, or refering to something from a different project. Google Desktop Search allows you to right-click on the found file and open the containing folder.

[‡] Download at *http://desktop.google.com.*

[§] Download at *http://desktop.google.com/plugins/i/indexitall.html.*

Copyrighted material

Spotlight in Mac OS X lets you do the same thing. When you find a file, if you hit Enter, it opens the file in the associated application. If you hit Apple-Enter, it opens the enclosing folder. Just as with Google Desktop Search, you can download Spotlight plug-ins that allow it to index your source files. For example, you can download a Spotlight plug-in from the Apple site to add Ruby code as an index target.[‖]

Spotlight now allows you to add search filters to your searches. For example, you can add *kind:email* to your search string and Spotlight will restrict its search to only emails. This harkens to the near future of searching, the ability to search via customizable attributes (see the next sidebar "The Near Future: Search Via Attributes").

THE NEAR FUTURE: SEARCH VIA ATTRIBUTES

Searching for files by title alone isn't that useful. Remembering the exact title is just as impossible as remembering where you put it. Searching by content is better because you are more likely to remember at least part of the file.

An even more powerful variant is coming to leading-edge software: the ability to search for files based on customizable attributes. For example, let's say you have some files that belong to the same project: Java source files, SQL schema, project notes, and tracking spreadsheets. Putting these items together in the file hierarchy makes some sense because they all relate to the same project. But what if some of the files need to be shared between several projects? Search allows you to find them based on the things in which they participate, not on their physical locations.

Eventually, we'll get filesystems that "understand" the idea that you can tag files with arbitrary attributes. You can do that now in Mac OS X using Spotlight Comments, which allows you to tag the files that belong to the same logical project without worrying about their physical location. Windows Vista also offers a similar feature. If your operating system offers this feature, use it! It is a much better way to organize your groups of files.

Find Hard Targets

NOTE
Try simple searching before resorting to "hard target" searching.

Google Desktop Search, Spotlight, and Vista's search are great for finding files when you know some of the content. But sometimes you want more sophisticated searching capabilities. None of the aforementioned tools supports regular expressions, which is a shame because regular

[‖] Download at *http://www.apple.com/downloads/macosx/spotlight/rubyimporter.html.*

Copyrighted material

expressions have been around a long time and offer an incredibly powerful search mechanism. The more efficiently you can find something, the sooner you can return your focus to the problem at hand.

All flavors of Unix (including Mac OS X, Linux, and even Cygwin in Windows) include a utility called *find*. Find is responsible for finding files from the given directory downward, recursing through directory structures. Find takes tons of parameters that allow you to refine your searches, including regular expressions for filenames. For example, here is a *find* invocation that locates all Java sources files with "Db" right before the file extension:

```
find . -regex ".*Db\.java"
```

This search tells *find* to start from the current directory (the ".") and find all files that have zero or more characters (the ".*") before the string "Db", which comes before the "." (which must be escaped because "." normally means "any single character"), followed by the file extension of Java.

find is pretty darn useful by itself, but when you combine it with *grep*, you have a really powerful team. One of the options on *find* is *-exec*, which executes the command(s) that follow with the option of passing the found filename as a parameter. In other words, *find* will find all the files that match your criteria and then pass each file (as it's discovered) to the command to the right of *-exec*. Consider this command (explained in Table 3-1):

```
find . -name "*.java" -exec grep -n  -H "new .*Db.*" {} \;
```

TABLE 3-1. Decoding the find command

Character(s)	What it's doing
`find`	Execute the *find* command.
`.`	From the current directory.
`-name`	Match the name of "*.java" (note that this isn't a regular expression, it's a filesystem "glob," where the * means all matches).
`-exec`	Execute the following command on each found file.
`grep`	*grep* command, the powerful *-nix utility used for searching for strings within files.
`-n`	Show line numbers of the matches.
`-H`	Show filenames of the matches.
`"new .*Db.*"`	Match the regular expression that states "all files with any number of characters, followed by the letters Db, then followed by any characters."
`{}`	Placeholder for the filename found by *find*.
`\;`	Terminate the command after -exec. Because this is a Unix command, you might want to pipe the results of this into another command, and the *find* command has to know when the "exec" is done.

Although this is a lot of work, you can see the power of this combination of commands (for two different but equivalent versions of this particular combination of commands, check out "The Command Line" in Appendix A). Once you learn the syntax, you can literally query your code base. Here is another, slightly more complex example:

Copyrighted material

```
find -name "*.java" -not -regex ".*Db\.java" -exec grep -H -n "new .*Db" {} \;
```

You can use this combination of *find* + *grep* during code reviews, in fact, this example comes from a query I did during a code review. We were writing an application that adheres to the typical layered application design, with a model, a controller, and a view layer. All the classes that accessed the database ended with "Db", and the rule was that you shouldn't construct those classes except in controllers. Issuing the *find* command allowed me to find out exactly where all the boundary classes were being constructed and nip the problem in the bud before someone did something wrong.

Here's another little command-line trick for finding stuff. What if you want to go to the directory where some application that's on your path lives? For example, say you want to temporarily go to the directory where the executable command *java* lives. You can do so with a combination of the *pushd* and *which* commands:

```
pushd `which java`/..
```

Remember, any command in backticks (the `character) executes before the rest of the command. In this case, the *which* command (which finds the location of applications on your path) finds where *java* lives. But *java* is an application, not a directory. Thus, we take that location and back up to the parent, *pushd*-ing to it. This is a great example of the *composability* of *-nix commands.

Use Rooted Views

A r*ooted view* is a view of a directory structure rooted at a particular subdirectory, where you see only the contents from the root directory downward. If you are working on a particular project, you don't care about the files from other projects. Rooted views allow you to eliminate the distraction of out-of-context files and focus just on the set of files upon which you need to work right now. All the major platforms support this concept, but the implementations are different.

Rooted Views in Windows

A rooted Explorer view ("rooted" on the *c:|work|sample code|art|art_emotherearth_memento* folder) appears in Figure 3-2.

This is a normal Explorer window, opened with the following parameters:

```
explorer /e,/root,c:\work\cit
```

The rooted view affects only this instance of Explorer. If you launch another Explorer window using typical means, you'll see a normal instance of Explorer. To take advantage of rooted views, create shortcuts to Explorer with the rooted view parameters above. Rooted views work in all versions of Windows, from Windows 95 through Windows Vista.

Copyrighted material

FIGURE 3-2. Rooted views in Windows

NOTE
Rooted views turn Explorer into a project management tool.

Rooted views are particularly good for project work and especially good if you use a file- or folder-based version control system (like Subversion or CVS). As far as the rooted instance of Explorer is concerned, your project files and folders make up the entire universe. You can access the plug-in Tortoise[#] (a Subversion management tool for Explorer) via any place you click within the rooted view. And, more importantly, you eliminate the distractions created by a bunch of folders and files that mean nothing to the project on which you are working.

Rooted Views in OS X

Root views work a little differently in Mac OS X. While you can't create a Finder view that is uniquely focused on a single directory structure like you can with Windows Explorer, you can still create specialized rooted views that cut through the massive directory structures. In Finder, you can create shortcuts to directories by dragging the directory in question to the sidebar or to the dock. This allows you to open that directory right from Finder, as shown in Figure 3-3.

[#] Download at *http://tortoisesvn.tigris.org/*.

Copyrighted material

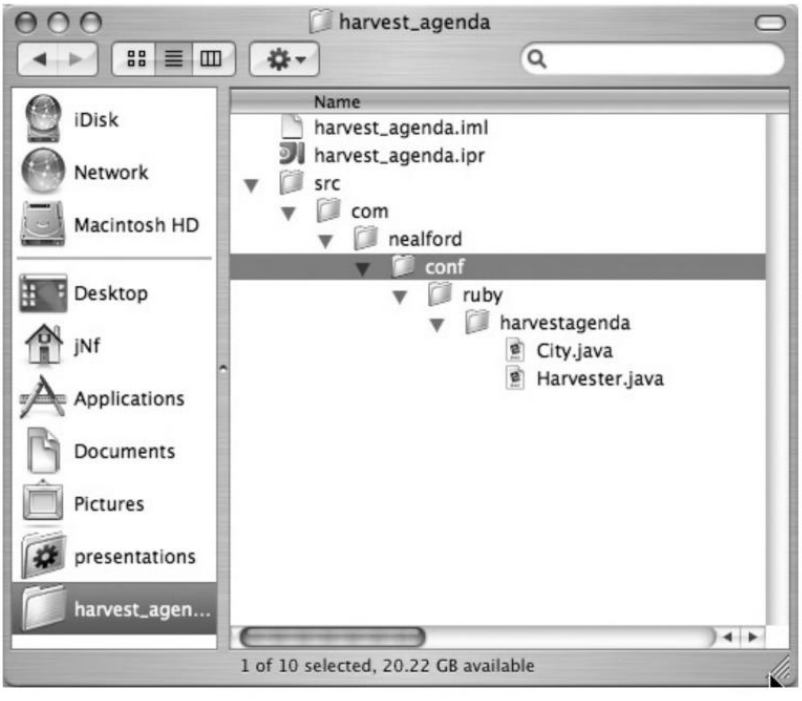


FIGURE 3-3. Rooted views in Finder

Use Sticky Attributes

The command line in Windows has a nasty surprise turned on by default called Quick Edit Mode. This is a switch set in the properties of the command window that allows you to select text to copy with the mouse. Any click inside the window starts a drag operation, highlighting a rectangular section of text, ready to copy it to the clipboard (oddly enough, the standard keyboard shortcut Ctrl-C doesn't work here; you have to hit the Enter key instead). Therein lies the problem. Because you are selecting text in the console window, it freezes all activity (that is, all the processes and threads occupying that window) as soon as you start dragging the mouse. This makes sense: it would be annoying to try to copy something that actively scrolls out of your way. Typically, in a windowed environment, it is perfectly safe to click anywhere in a window to give it focus. But if you click in a command prompt window, you've inadvertently started a drag operation, which freezes all the processes. Ironically, setting focus to a window can accidentally destroy the focus on the job at hand because you start wondering "Why is nothing happening in that window?" You can fix this "feature" using *sticky attributes*.

Copyrighted material

NOTE
Take advantage of built-in focus facilities (like colors).

Windows keeps track of customized settings in command prompts via the title of the window. When you close a command prompt in Windows, it asks if you want to save these settings for all command prompts with the same title (thus making them "sticky"). You can leverage that to create specialized command prompts. Create a shortcut to launch a window with a specific title, set some options, and save the options for the window as you close it. For development work, you need a command prompt with the following characteristics:

- Almost infinite scrolling. The default is a measly 300 lines, which will easily scroll off when doing something interesting. Set it to 9999 lines (and ignore the 1990s-era warning that your command prompt will now eat a precious 2 MB of memory).
- The widest width that your screen will support without horizontal scrolling. Reading wrapped lines in a command prompt is tedious and error prone.
- Set the position. If this is a command window with a single purpose (like a servlet engine or Ant/Nant/Rake window), have it always appear in a known location. You will learn that location quickly, allowing you to know what this command prompt does without even looking at it.
- Set unique foreground and background colors. For common command prompts (like servlet engines), the color becomes an important clue as to the purpose of the window. You can quickly identify that a cyan background with yellow text is a Tomcat window, while a blue background with green text is a MySQL prompt window. When you cycle through open command prompts, the color (and position) tell you the purpose of this window faster than you can read.
- And, of course, turn off Quick Edit Mode.

Use Project-Based Shortcuts

All the major operating systems have some alias, link, or shortcut mechanism. Use it to create project management workspaces. Frequently, you have projects that have documents strewn all over your hard drive: requirements/use cases/story cards in one place, source code in another, database definitions in another. Navigating between all those folders is a waste of time. Instead of forcing all your files for a project into a single location, you can put them all together virtually. Make one project-based folder that has shortcuts and links for the entire project. You'll find that you spend much less time spelunking around the filesystem.

NOTE
Use links to create virtual project management folders.

Copyrighted material

Place your project management folder under one of the Quick Launch buttons in Windows or on the dock in Mac OS X. These two areas don't support a large number of items, but using them for just a few project consolidator folders makes sense.

Multiply Your Monitors

Monitors have gotten cheap, and developers can use the extra real estate. It is penny-wise and dollar foolish not to supply developers with ultra-fast computers and dual monitors. Every moment that a knowledge worker stares at an hourglass is pure wasted productivity. Having to work to manage all the overlapping windows on a cramped monitor also wastes time.

Multiple monitors allow you to write code on one and debug on the other. Or keep documentation alongside your coding. Having multiple monitors is just the first step, though, because you can also segregate your dual workspaces into a bunch of specialized views using virtual desktops.

Segregate Your Workspace with Virtual Desktops

> NOTE
> **Virtual desktops unclutter your stacks of windows.**

One of the cool features from the Unix world is *virtual desktops*. A virtual desktop is just like your regular desktop, with windows in a certain arrangement, but the "virtual" part indicates that you can have more than one. Instead of having one massively jumbled desktop with your IDE, your database console, and all your email, instant messaging, browsers, etc. on it, you can have singly purposed desktops for each logical grouping of activities. A massive pile of windows on your desktop distracts you from your focus because you constantly have to sort through the windows.

Virtual desktops used to exist solely on high-end Unix workstations (which had the graphical horsepower to support such a thing). But now they exist on all the major platforms. On Linux, both GNOME and KDE have virtual desktops built-in.

The Leopard version of Mac OS X (version 10.5) added this feature, called Spaces. But previous Mac OS X users aren't left out: several open source and commercial virtual desktops exist, such as VirtueDesktops.* It offers sophisticated features such as "pinning" applications to a certain desktop (meaning that application will appear only on that desktop, and the focus will change to that desktop if you select that application). This is a great feature for developers, who typically set up certain desktops for specific purposes (development, documentation, debugging, etc.).

* Download at *http://virtuedesktops.info/*.

Copyrighted material

FIGURE 3-4. Managing desktops with Virtual Desktop Manager

One of my recent projects was a Ruby on Rails gig where we were pair-programming on Mac Minis (the little smaller-than-a-breadbox machines that you buy without a monitor or keyboard). They make surprisingly good development machines, especially with two keyboards, mice, and monitors. What made them great environments, though, were the virtual desktops. We had each machine set up the same way (to preserve sanity as we swapped pairs), with all the development tools on one desktop, documentation on another, and the running application (a terminal window running the web server in debug mode and browser) on a third. Each desktop was completely self-contained, and as we switched applications, the appropriate desktop rotated into view. The environment allowed us to leave all the windows for a particular desktop in the same place all the time, with very little tiling and cascading. In my last gig, I was coding alone on Windows, but I still set up "communication," "documentation," and "development" desktops, which cut down on clutter and aided my sanity.

Copyrighted material