
This was a talk presented at DevWorld 08 in Melbourne, Australia.  Thank you to the AUC and the DevWorld organisers for inviting me to speak there.


# The Business of Development

AUC

/dev/world/2008
Rydges Melbourne

André Pang

Realmac Software

I assume that you want to code professionally; i.e. as your main profession. If you're a hobbyist, all of the advice here applies to you too, but how much you want to heed it really depends on how seriously you take your project and how much time you can invest.

(Also, NSNoFoodError!)

## Trism

USD$250,000 in two months

## Delicious Library

USD$250,000 in one month

## RapidWeaver

:)

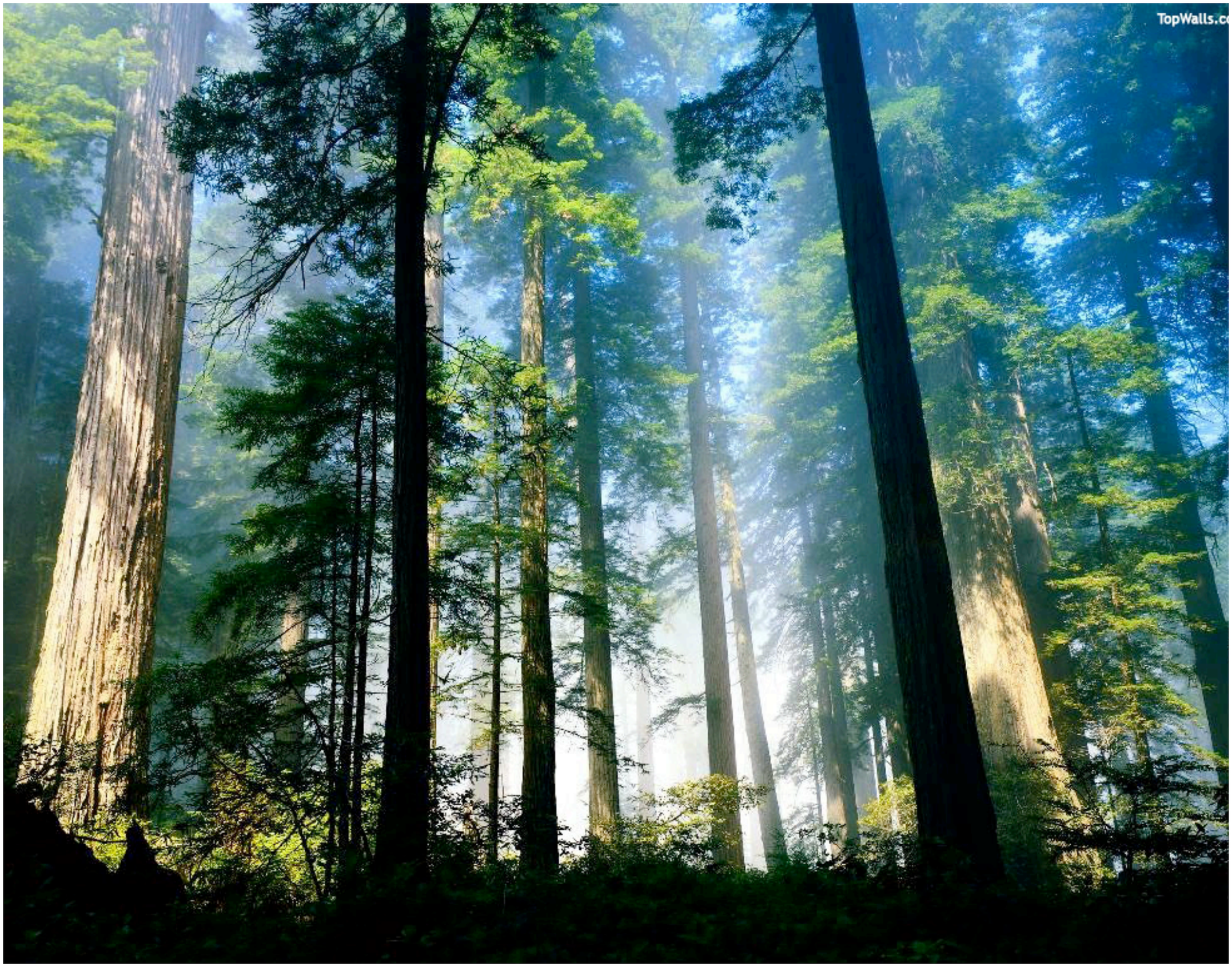
Three leading applications. (Trism is for iPhone). I'll be mum about RapidWeaver sales (and, truth be told, I don't know what they are anyway), but it's enough to keep six of us employed at the moment. You can do the maths.

This talk is not about turning you into this guy.  You are a coder, not a businessperson.  (You may be both, but deep down I'm guessing you still consider yourself an engineer.)

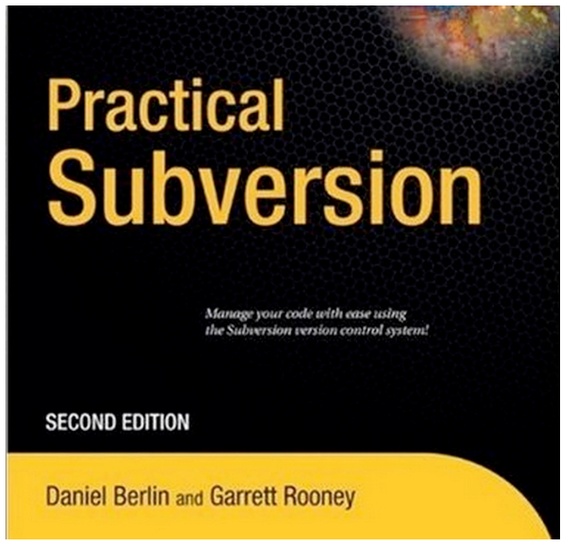

You are this guy: an essential part of a bigger team.  The goal here is to teach you to think about how you can support your workmates better and understand their workflows, so that your combined output with someone else is 220% rather than 150%.


# Context

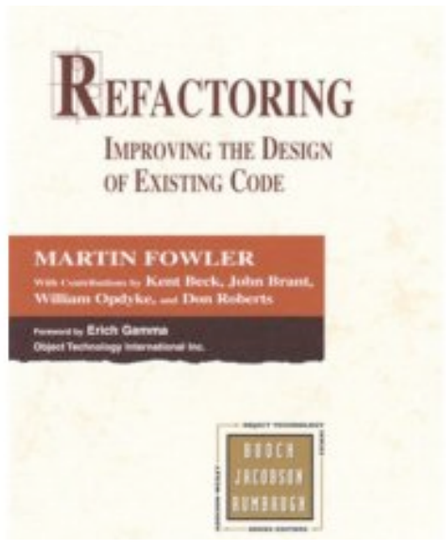
I’ll be spinning tales from my experience with cineSync and RapidWeaver.




The first goal: sustained growth.  More specifically, sustained growth of your code, and things you can do to make your life easier as a developer when working with non-developers on your team.



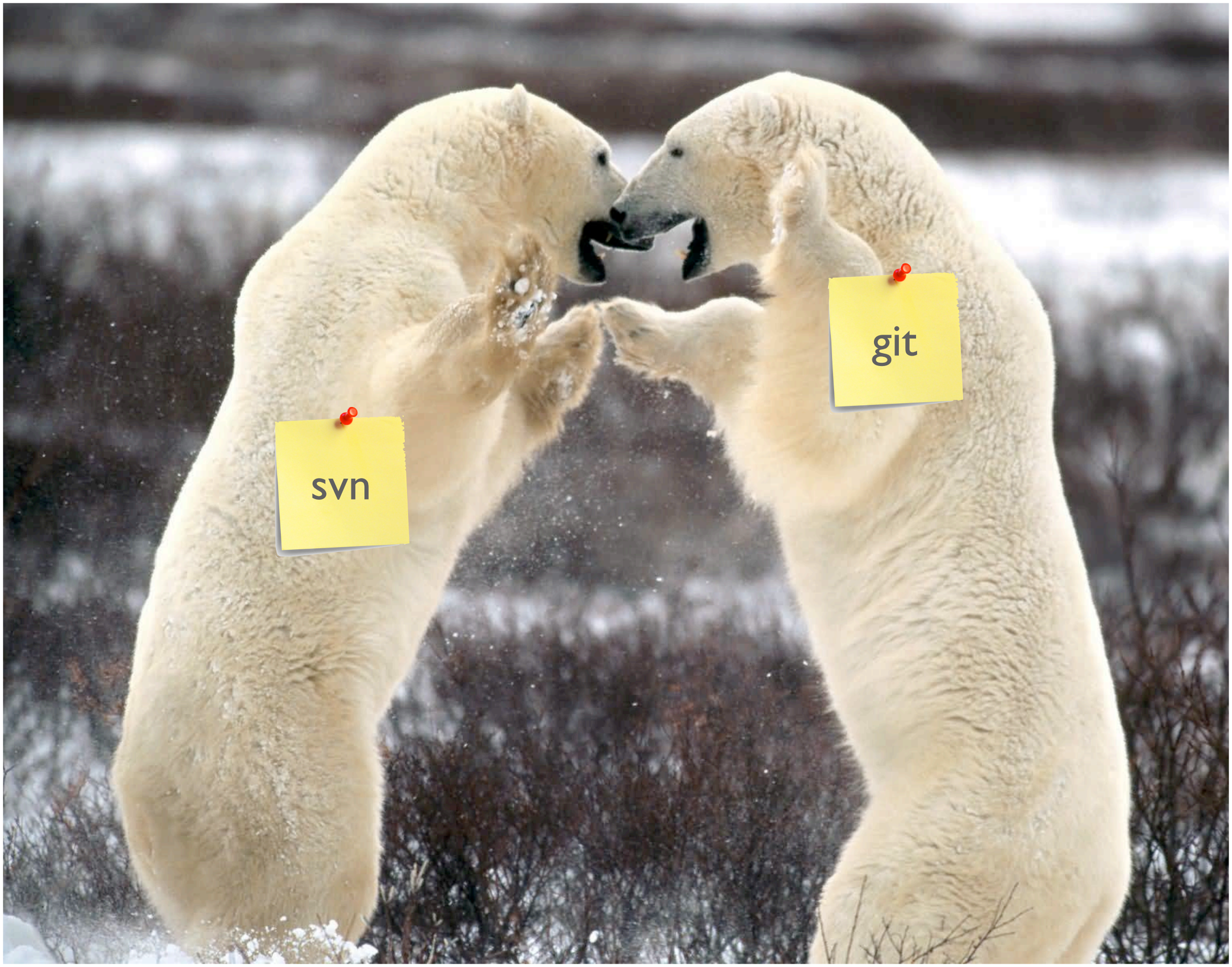


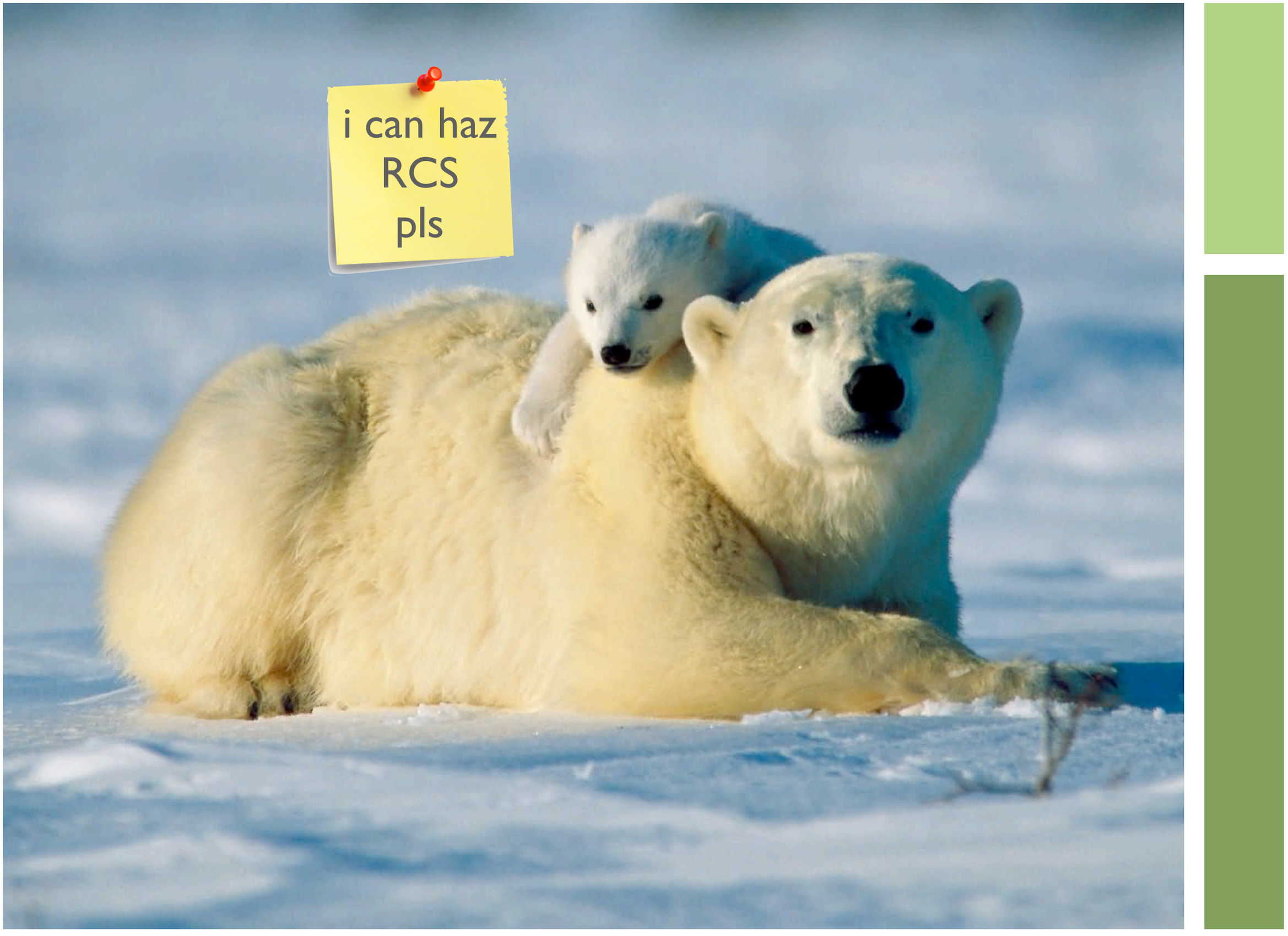

trac
Integrated SCM & Project Management

Buildbot

The good news is that “best practices” will get you a very, very long way to making good code. Good code is simply good communication to whoever’s reading the code. Good practices extends this communication from you to the rest of the team, instead of from the code to you.




Revision control systems are absolutely essential.  If you are not using one, I have no sympathy for you.  But there's a ton of them out there: centralised ones such as Subversion, CVS, Perforce, and a plethora of decentralised ones such as darcs, git, Bazaar, and Mercurial.  Which one do you use?

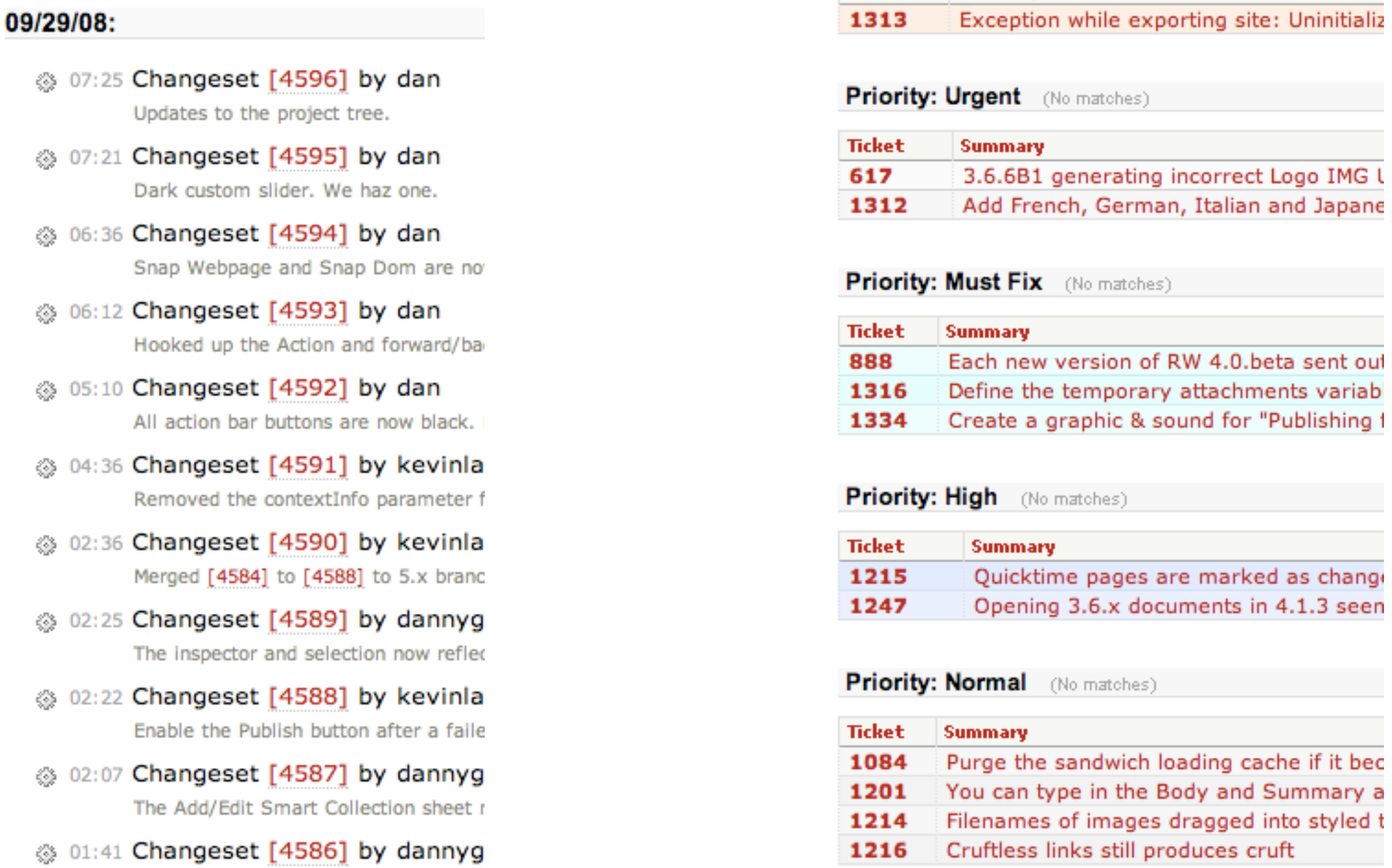



Answer is: just pick one and run with it.  I personally recommend Subversion since it has the best tool support out there, and if you're working with non-developers, it has a wide range of excellent clients available across many platforms, and even has distributed clients available for the geeks out there (such as git-svn).  It's relatively weak with merging (even compared to Perforce), but it's adequate for most users.  Something is infinitely better than nothing.


# Nightly Builds

Do nightly builds. Apart from making sure that your trunk always compiles with no errors and that it actually works, your workmates can simply download the nightly build so that they're working off the same version that you're working on. Ultimately, you want to set up builds so that you can run a build script to automatically check out and build a proper release, rather than you building it on your home machine. This guarantees consistency and that you have things such as debug symbols available for each release.

Make sure that you have one branch (usually the trunk in Subversion) that can be made available for release immediately. If you discover a mission-critical bug at 4pm today, you want to be able to move quickly. You do not want to fix up 10 compile errors when you need to do a release.

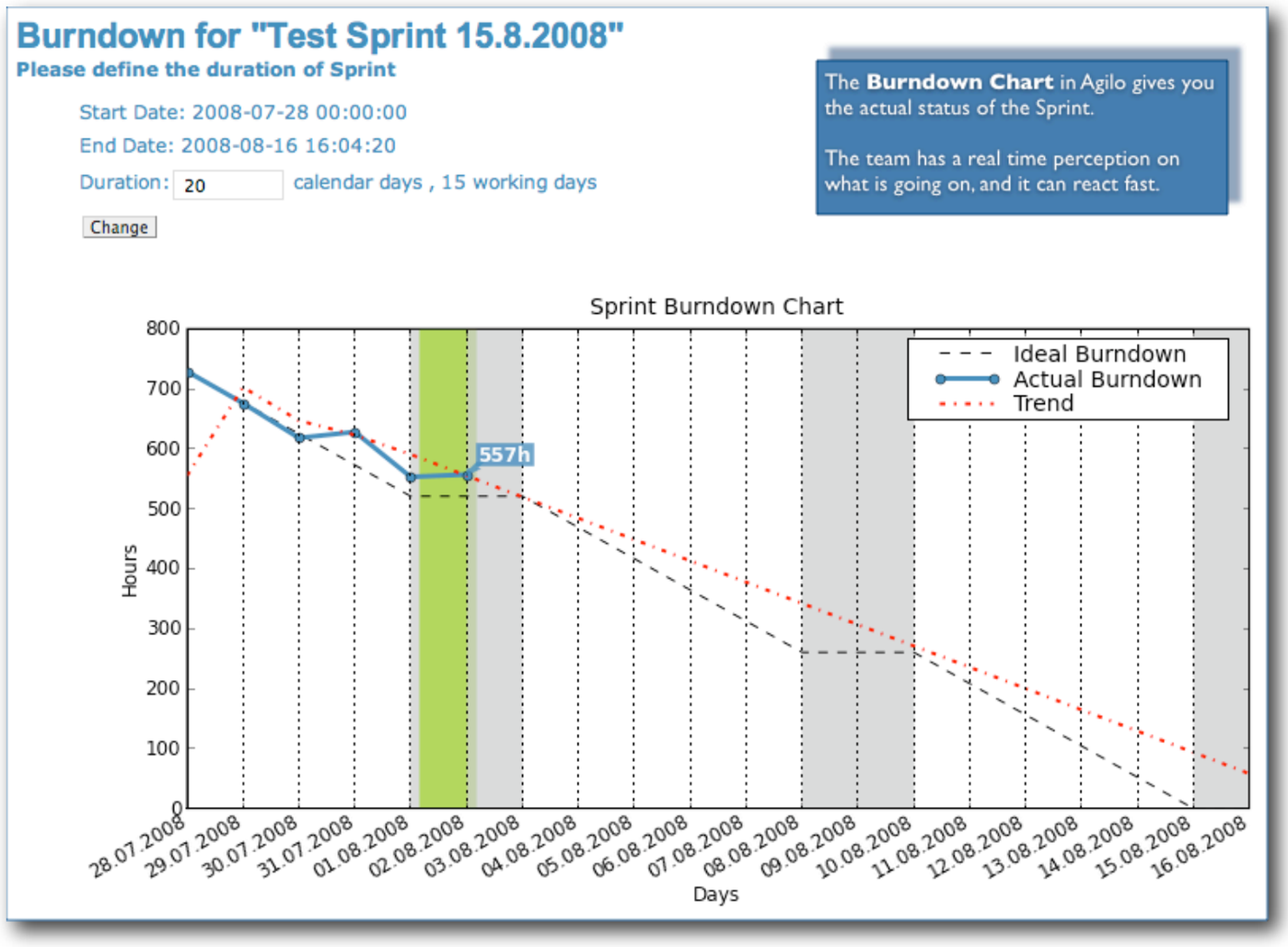



# Trac

Get a bugtracker. I recommend Trac: it's a little painful to set up, but it's free, and its Subversion integration with the timeline and source code browser makes your progress easier to track for non-developers.

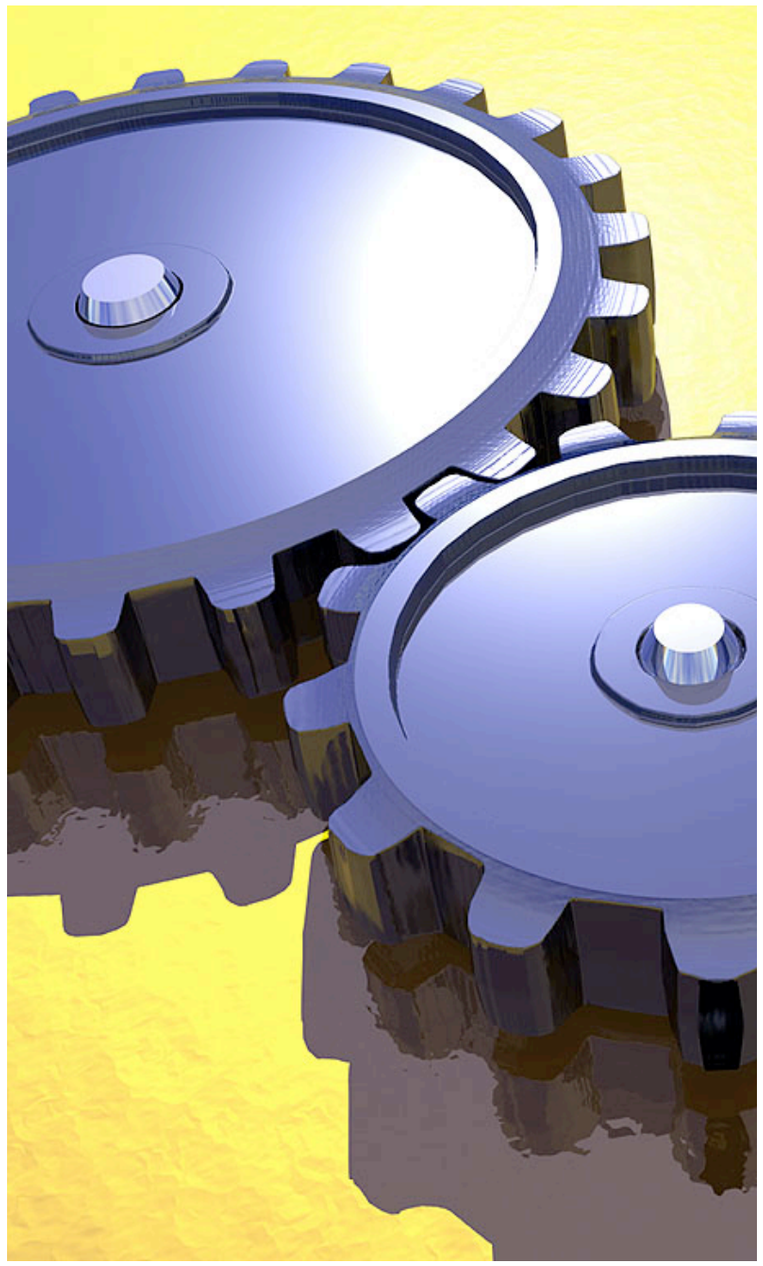
Plus, you get a Wiki for free. Put all your developer-related notes and documentation on your Wiki. Your mission is to get your co-workers to say "hey, how does x work… oh hey, I'll just go check the Wiki".

# Burndown Charts




Do time estimates. Many bug trackers will have a field for a ticket where you can keep track of the estimated number of hours for a task, and the actual number of hours spent on it. There's a burndown plugin available for Trac to help you with this. FogBugz does evidence-based estimation to give you an estimated ship date based on your progress.

Your product manager can look at this chart and decide to change plans for your next milestone depending on how things are going.


# Just Three Variables

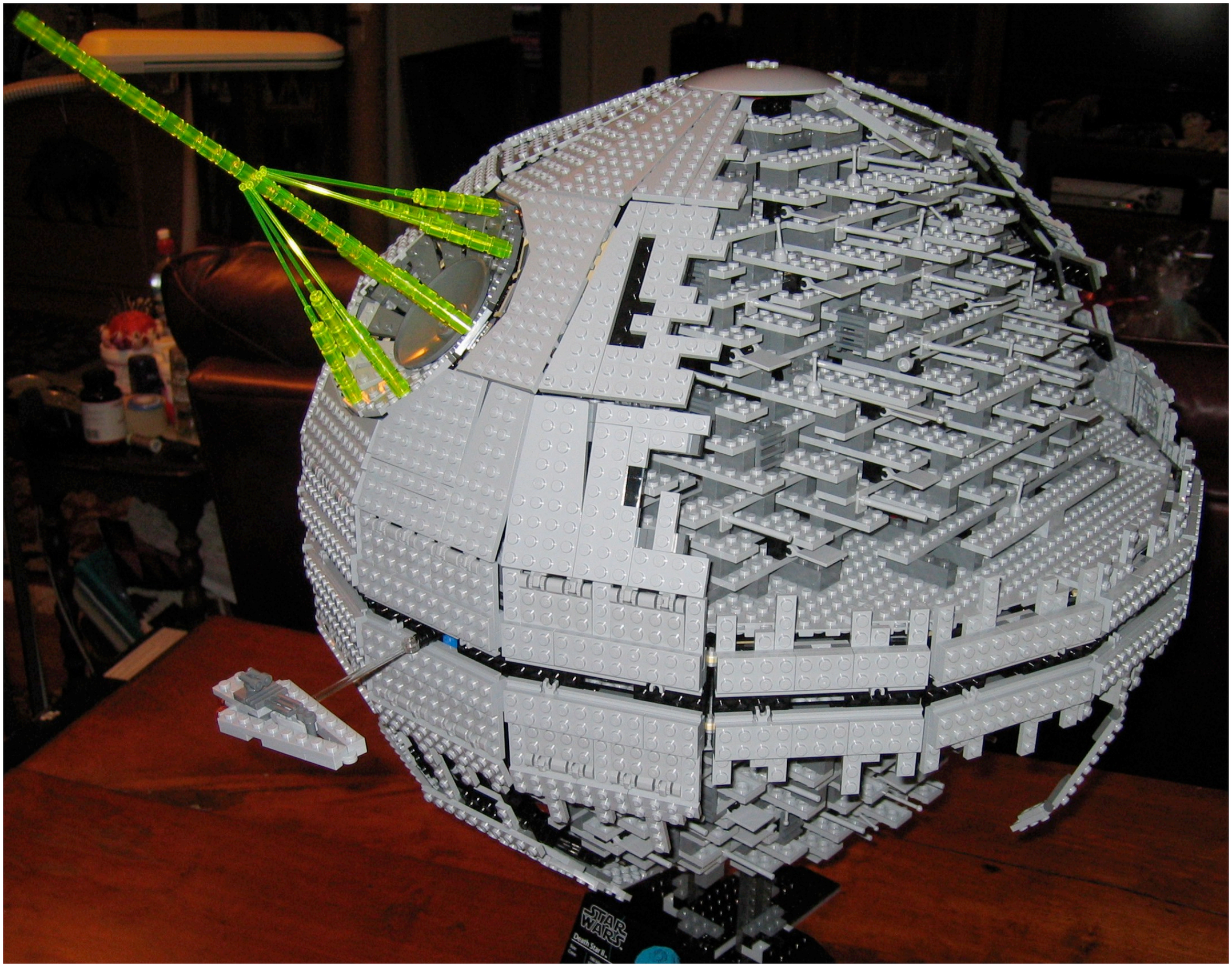
There are only three variables you can adjust when deciding on a release: time, features, and quality. Fix one or two of those variables, and you need to change the other. Want to release tomorrow with a product that's stable? You need to pull features. Need to release soon with this one really essential feature? Your quality will suffer.

Your product manager, not you, is responsible for deciding how important these three variables are. Your bug tracker and burndown charts give them essential information on tracking your progress.


# TPS Reports: Do Not Want

Revision control, source code browsing and timelines, and nightly builds are all a way to keep the communication channels open with your “product manager” (which may or may not be you), who is ultimately in charge of the product.  They don’t have to check with you to see what the status of your code is; they can see it for themselves.  This makes them happy because they know what’s going on, and this makes you happy because you do not have to fill out TPS reports.

Mac OS X comes with a huge number of very useful technologies: Core Image, Core Animation, Core Data, QuickTime.  While they certainly can make coding easier, do they make sense from a business point of view?



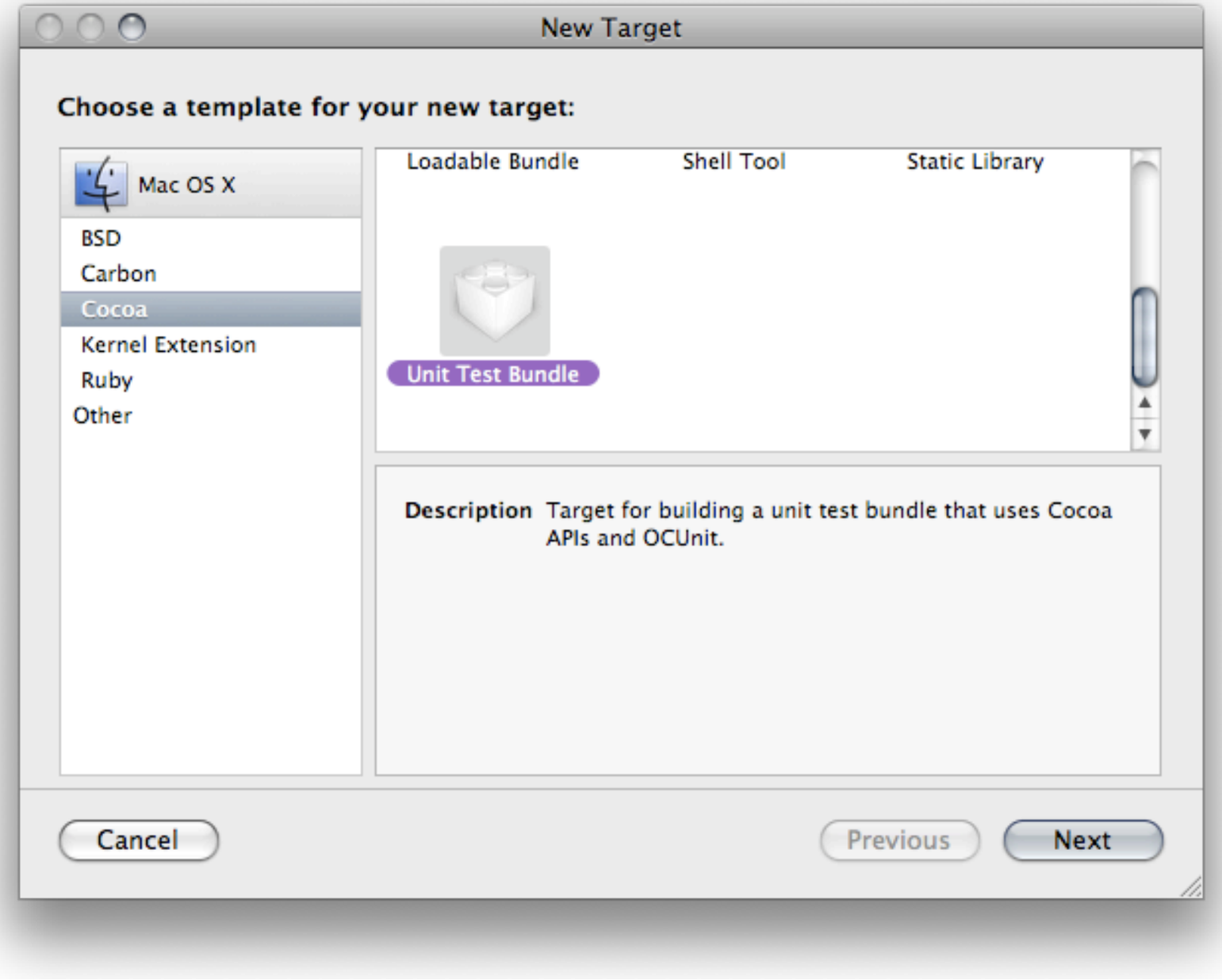

We've traditionally been taught that code re-use is extremely important, and to build complex systems out of smaller understandable components.  Why re-invent the wheel if a library out there (perhaps an open-source one) does the job for us?

“If it's a core business function—do it yourself, no matter what.”

— Joel Spolsky

I like this quote from Joel Spolsky. If something’s part of the core functionality of your application, you want to have full control over it. Maybe you don’t have to write it all yourself, but you probably really, really want at least the right to make full modifications to the source code without being beholden to someone else. I personally do not use Core Data for this reason (I consider the data model to be completely fundamental to your application.)

For cineSync, there were no decent BSD-licensed C++ Jabber/XMPP libraries around when we wrote it (and there probably still aren’t). We could’ve used an existing BSD-licensed C library, but really wanted the extra abstractions that C++ gave us because it fit in with the rest of our code base, so we wrote it ourselves. For RapidWeaver, exporting rich text to HTML is one of its core abilities, so we wrote that ourselves instead of relying on AppKit’s rich text to HTML conversion.




# Test Complex Systems

You do not want customers yelling at you because of regressions (breaking things that used to work OK). Testing is the best way I know of to ensure regressions don't happen: do unit testing, and make it automated otherwise you'll never run it. Make it part of your nightly build scripts.

As a side benefit, testable code is generally well-designed code. Making code testable is much harder than you think, and is a good indicator that you're following MVC principles. Martin Fowler goes so far as to define legacy code as "code without tests", and his entire mission in modernising legacy code is to make it testable and to create a test suite, so that you can make sweeping changes with confidence.


# Cross-Platform

Consider making your code cross-platform: have versions available for Mac, Windows and Linux. This is a no-brainer from the business perspective: the Mac market may be 20-30 million users, but the Windows market is stupidly larger than that, which means extra revenue. (The shareware conversion rate with Windows may or may not be lower, but the sheer volume of users makes up for that.) As with testing, a side bonus is that cross-platform code forces you to design your code better, and have proper abstractions and hooks in place for the various platforms you're targeting. Depending on what your product is, targeting Linux users may also make a lot of sense.



# Also Cross-Platform

If you're not going to be cross-platform for the Mac, Linux and Windows, definitely consider going cross-platform for the Mac and iPhone. You probably want to do this for fun anyway, and could possibly have a very nice revenue stream out of it. The Omni Group makes a lot money from OmniFocus for iPhone as well as the Mac version.

And, once again, you reap the benefits of the better design that this will bring you: you can guarantee that your model code doesn't have any View/Controller code in there since it should only be using the Foundation framework, rather than Foundation & AppKit or Foundation & UIKit.

YAHOO! TECH Groups

Sign In
New User? Sign Up

Tech - Groups - Blog - Hel

**Save time and money on popular products!**
Search millions of products from over 100,000 stores. Instantly compare prices, shopping.yahoo.com

**Yahoo! Personals - Search Our Ads - Free**
We know tons of great, cute singles. The only thing missing is you. Try it now, personals.yahoo.com

**Domain Sales**
Yahoo! Domains Only $1.99/1st yr. Includes 24x7 support, email & more.
domains.yahoo.com

**macsb** · Macintosh Software Business

Search for other groups... Search

Home

**Members Only**
Messages
Post
Files
Photos
Links
Database
Polls
Members
Calendar
Promote

Info Settings

**Group Information**
Members: 2085
Category: OS X
Founded: Jan 29, 2004
Language: English

**Stay up to speed on the latest Groups news and updates, visit the Groups blog today!**

## Home

Join This Group!

**Activity within 7 days:** 10 New Members - 89 New Messages - New Questions

**Description**

This group is for small, independent Macintosh developers who want to talk with other developers about the business of Mac development. Questions on pricing, packaging, advertising, e-commerce providers, and so on are on-topic. Note that this list isn't a vehicle for promotion: announcements and press releases are off-topic.

**Message History**

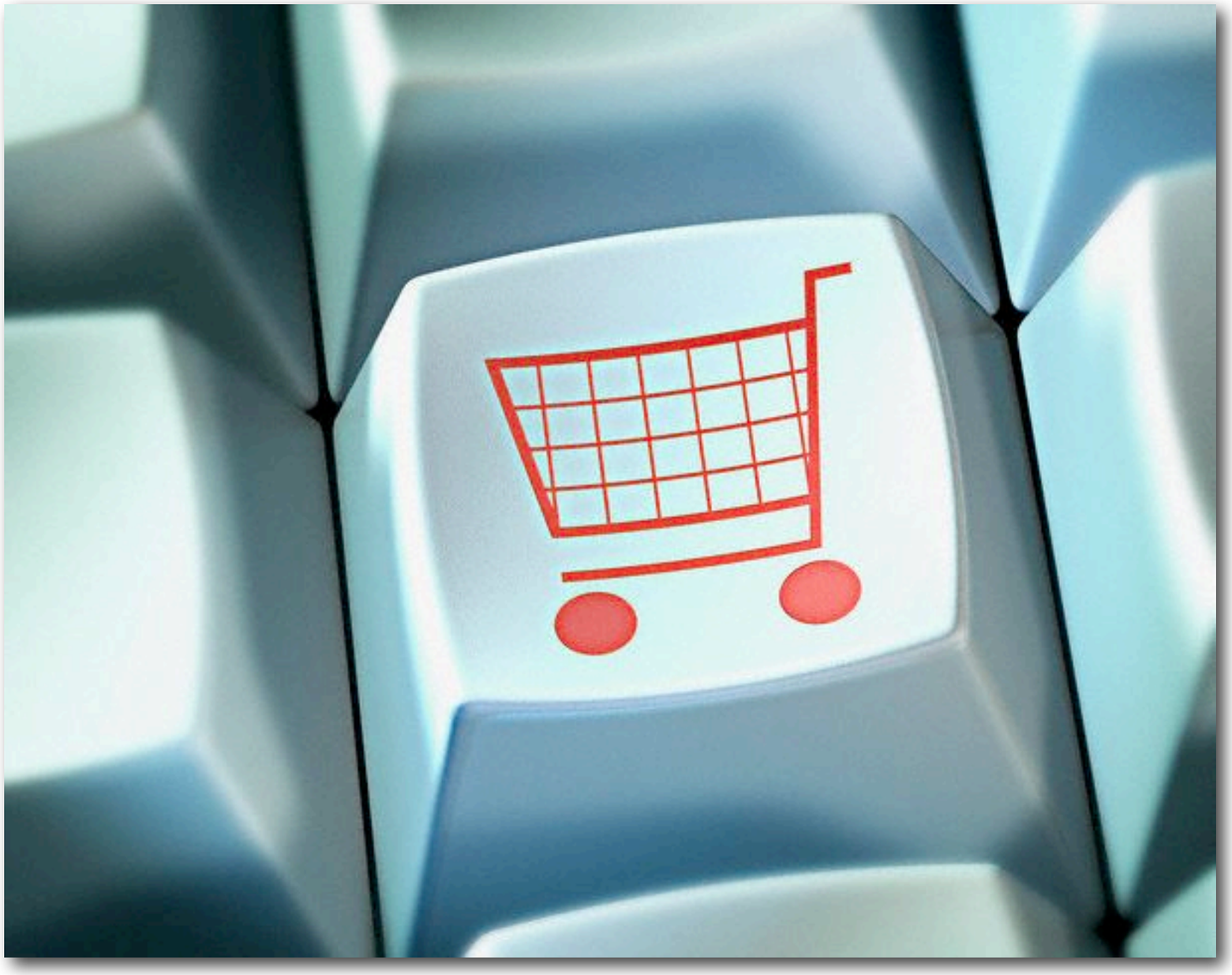
| | Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2008 | 273 | 384 | 568 | 188 | 239 | 182 | 234 | 116 | 208 | | | |
| 2007 | 352 | 442 | 294 | 685 | 281 | 382 | 327 | 531 | 303 | 403 | 449 | 273 |
| 2006 | 235 | 264 | 280 | 273 | 277 | 257 | 197 | 272 | 459 | 315 | 385 | 244 |
| 2005 | 92 | 76 | 101 | 97 | 136 | 423 | 131 | 152 | 175 | 141 | 97 | 179 |
| 2004 | 116 | 193 | 99 | 95 | 39 | 138 | 16 | 6 | 33 | | | 13 |

For more information on general business practices, check out the MacSB mailing list <http://tech.groups.yahoo.com/group/macsb/>, which has been around for several years, is reasonably low-traffic, and often has interesting threads.

You will eventually have to deal with support.  With every feature you add and every bug you fix, think about the support load that it'll cost you.  We have omitted some very cool features from RapidWeaver specifically because we knew it would be a support nightmare.


Everybody hates licensing (except maybe lawyers). Users hate it because it gets in their way, and developers hate it because it encumbers their product and they have to write all this extra code to deal with it. Don't spend more than a day or two on this crap. Keep the registration scheme simple, because otherwise you'll have users emailing you by the truckload with the most inane licensing questions when your beautiful complexity fails to work for some very obscure reason. Have automated ways of retrieving lost license keys or you will be wasting time that could be spent coding instead.


There are a few free licensing frameworks out there; check out AquaticPrime <http://aquaticmac.com/> if you're looking for something.  I haven't used it personally, but a few other folks have, and I haven't heard too many complaints about it.

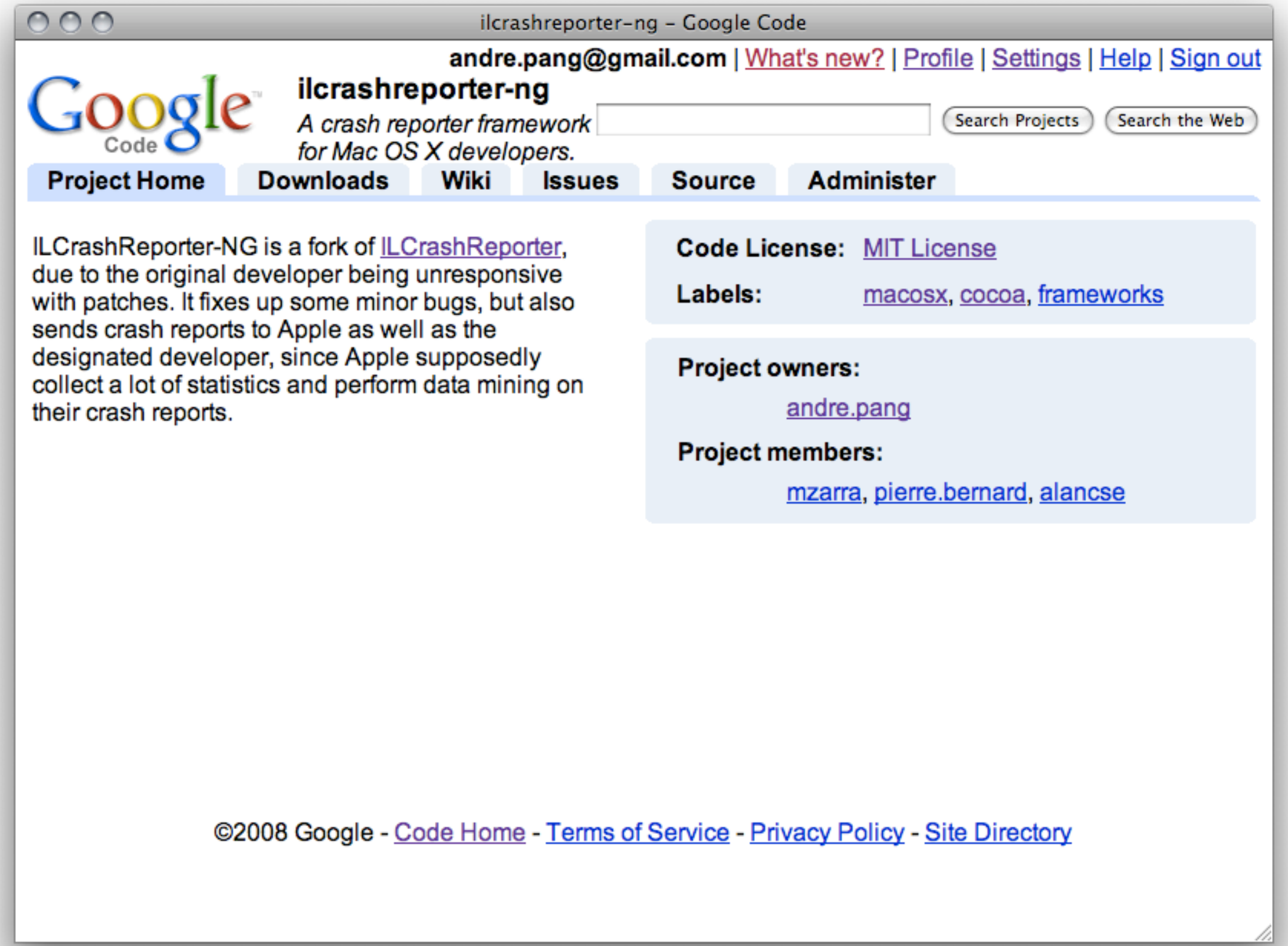


There are a ton of online merchants/commerce providers out there: eSellerate, Kagi, Paypal, Potion Factory Store, etc. I'd recommend eSellerate when you start: the ~15% cut sounds like a lot, but you get a reasonable amount of goodies in return for that, e.g. a licensing scheme, and a library you can drop into your application that does registration all inside the app. Later once you're more profitable, you may want to switch to another merchant provider that takes a lower cut.


Design a webpage that you're proud of. This is Realmac Software's homepage <http://realmacsoftware.com/>, and it was obvious that it wasn't designed by a programmer with no aesthetic sense like myself. Your Web presence is the most dauntingly important image that you project out to absolutely everybody. People are going to be far less attracted to your software if your webpage doesn't inspire them.


This is the webpage for cineSync <http://cinesync.com/>: it's not as pretty as the Realmac Software site, but we didn't have the luxury of having people on our team who were Web design geniuses who did it for a living. Nevertheless, it's simple and functional enough, and at least looks good.

Also note the cool logo, which doubled as the cineSync application icon. We did pay a good sum of money for the logo, but it was well worth it. Your application icon is your application branding, and is even more important than your website.

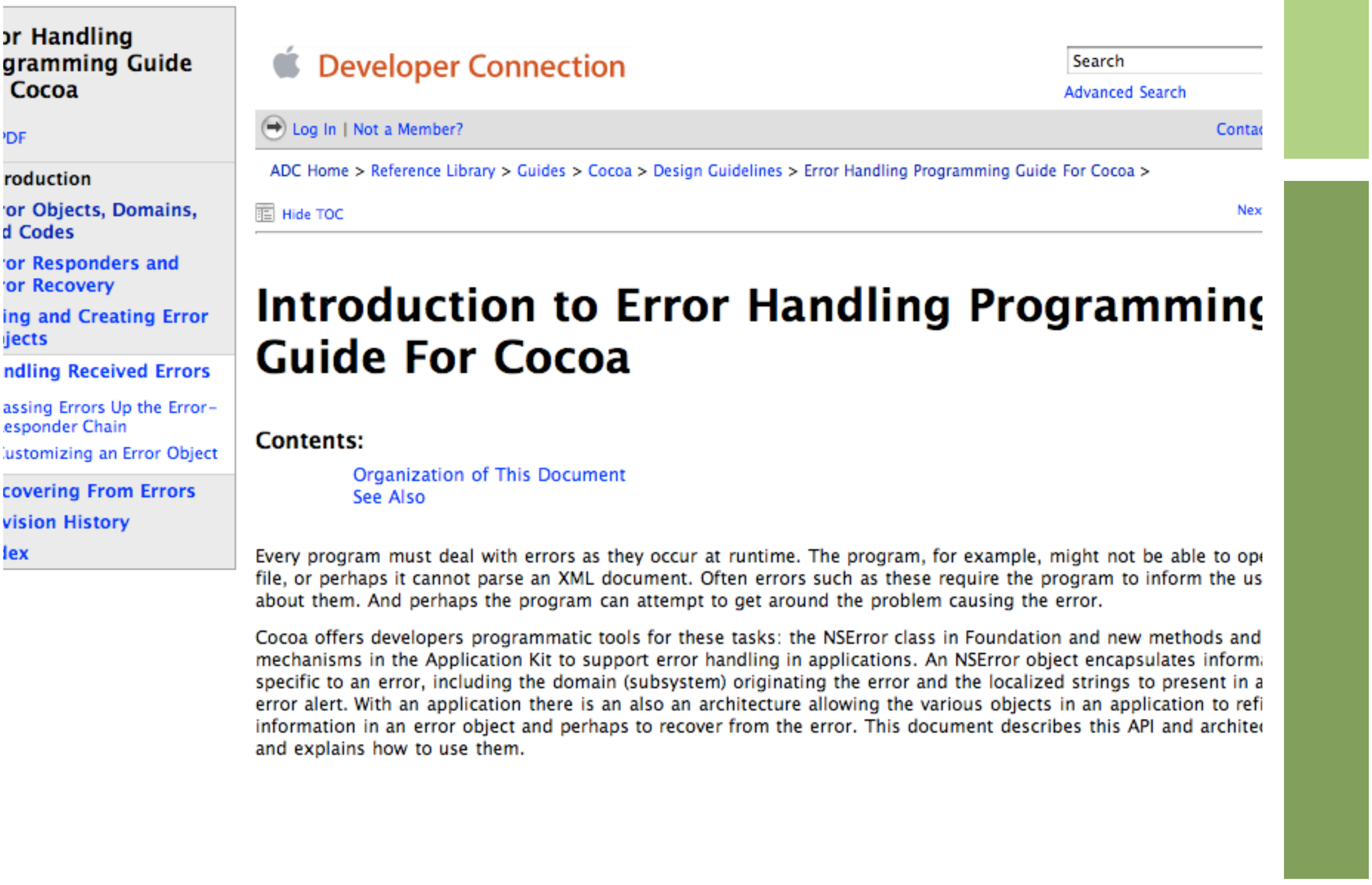



Use a crash reporter of some sort, otherwise every single crash that happens out in the field (and there will be plenty of them) will never get noticed.  Apple's own crash reporter is great, but unfortunately never sends the data to anyone besides Apple.  I'm bias since I worked on ILCrashReporter-NG <http://code.google.com/p/ilcrashreporter-ng>, but there are other open-source ones out there that are equivalent.  Pick one, any one, and use it.

When we put a crash reporter into RapidWeaver, we had dozens of crash reports coming back every week, and it was depressing. It was, however, more depressing that this was already happening and we didn't know about it: at least now we had the power to fix it.  Even if only 5% of crash reports give you useful information, you'll slowly build up more stable software over time.


Use an automatic application updater of some sort. Sparkle <http://sparkle.andymatuschak.org/> is very popular and even gathers anonymous statistics about your users, and Google has Update Engine <http://code.google.com/p/update-engine/> if you don't like Sparkle for some reason. Both are very simple to embed into your application.

You want an automatic application updater to reduce support load. If you make it easy for your customers to stay up-to-date with the latest version of your software, you'll get a lot less bug reports related to old versions that may have the bug fixed in the latest version.



or Handling
gramming Guide
Cocoa

DF

roduction
or Objects, Domains,
d Codes
or Responders and
or Recovery
ing and Creating Error
jects
ndling Received Errors
assing Errors Up the Error-
esponder Chain
ustomizing an Error Object
covering From Errors
vision History
lex

Developer Connection

Search
Advanced Search

Log In | Not a Member?
Contac

ADC Home > Reference Library > Guides > Cocoa > Design Guidelines > Error Handling Programming Guide For Cocoa >

Hide TOC
Nex

# Introduction to Error Handling Programming Guide For Cocoa

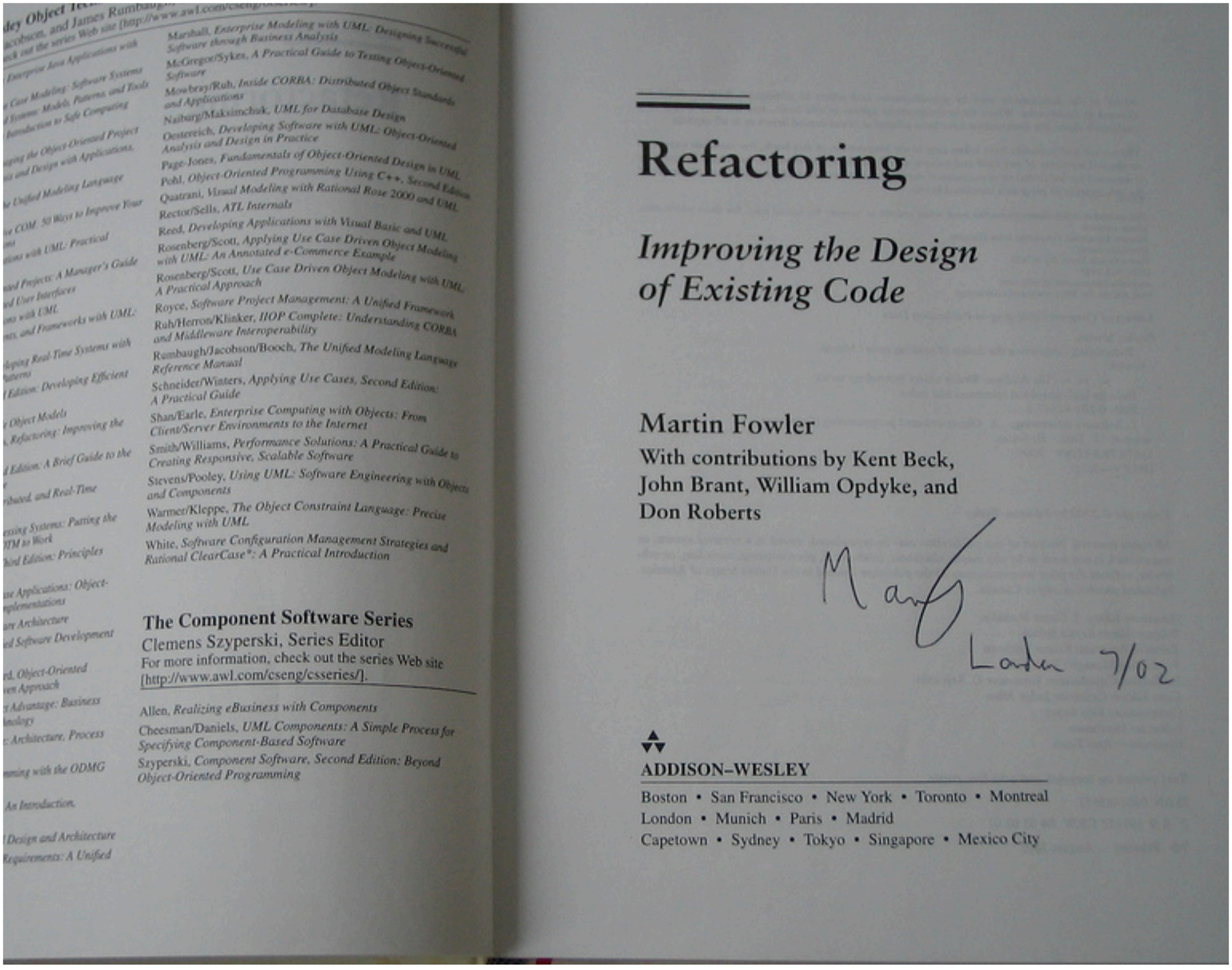
**Contents:**



Every program must deal with errors as they occur at runtime. The program, for example, might not be able to ope file, or perhaps it cannot parse an XML document. Often errors such as these require the program to inform the us about them. And perhaps the program can attempt to get around the problem causing the error.

Cocoa offers developers programmatic tools for these tasks: the NSError class in Foundation and new methods and mechanisms in the Application Kit to support error handling in applications. An NSError object encapsulates informa specific to an error, including the domain (subsystem) originating the error and the localized strings to present in a error alert. With an application there is an also an architecture allowing the various objects in an application to refi information in an error object and perhaps to recover from the error. This document describes this API and architec and explains how to use them.

Cocoa has a rich error handling mechanism—NSError—that is often daunting at first. Error handling is a hard part of programming, but if you don't get it right, you'll be paying for it with more support. Your customers will often be running your software on slightly broken systems or in completely unexpected environments, and when they do, all those excruciating, rare errors that could "never possibly happen" can and will happen. If you don't present a friendly error message that contains a helpful recovery suggestion, you're just going to have to deal with it later in your support inbox. Multiply this by the number of users you have, and that one hour you spent skipping out on dealing with errors correctly is going to seem like nothing at all later.



*Marshall, Enterprise Modeling with UML: Designing Successful Software through Business Analysis*
*McGregor/Sykes, A Practical Guide to Testing Object-Oriented Software*
*Mowbray/Ruh, Inside CORBA: Distributed Object Standards and Applications*
*Naiburg/Maksimchuk, UML for Database Design*
*Oestereich, Developing Software with UML: Object-Oriented Analysis and Design in Practice*
*Page-Jones, Fundamentals of Object-Oriented Design in UML*
*Pohl, Object-Oriented Programming Using C++, Second Edition*
*Quatrani, Visual Modeling with Rational Rose 2000 and UML*
*Rector/Sells, ATL Internals*
*Reed, Developing Applications with Visual Basic and UML*
*Rosenberg/Scott, Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example*
*Rosenberg/Scott, Use Case Driven Object Modeling with UML: A Practical Approach*
*Royce, Software Project Management: A Unified Framework*
*Ruh/Herron/Klinker, IIOP Complete: Understanding CORBA and Middleware Interoperability*
*Rumbaugh/Jacobson/Booch, The Unified Modeling Language Reference Manual*
*Schneider/Winters, Applying Use Cases, Second Edition: A Practical Guide*
*Shan/Earle, Enterprise Computing with Objects: From Client/Server Environments to the Internet*
*Smith/Williams, Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*
*Stevens/Pooley, Using UML: Software Engineering with Objects and Components*
*Warmer/Kleppe, The Object Constraints Language: Precise Modeling with UML*
*White, Software Configuration Management Strategies and Rational ClearCase®: A Practical Introduction*

**The Component Software Series**
Clemens Szyperski, Series Editor
For more information, check out the series Web site [http://www.awl.com/cseng/csseries/].

*Allen, Realizing eBusiness with Components*
*Cheesman/Daniels, UML Components: A Simple Process for Specifying Component-Based Software*
*Szyperski, Component Software, Second Edition: Beyond Object-Oriented Programming*

# Refactoring

## *Improving the Design of Existing Code*


**Martin Fowler**
**With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts**



**ADDISON–WESLEY**
Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

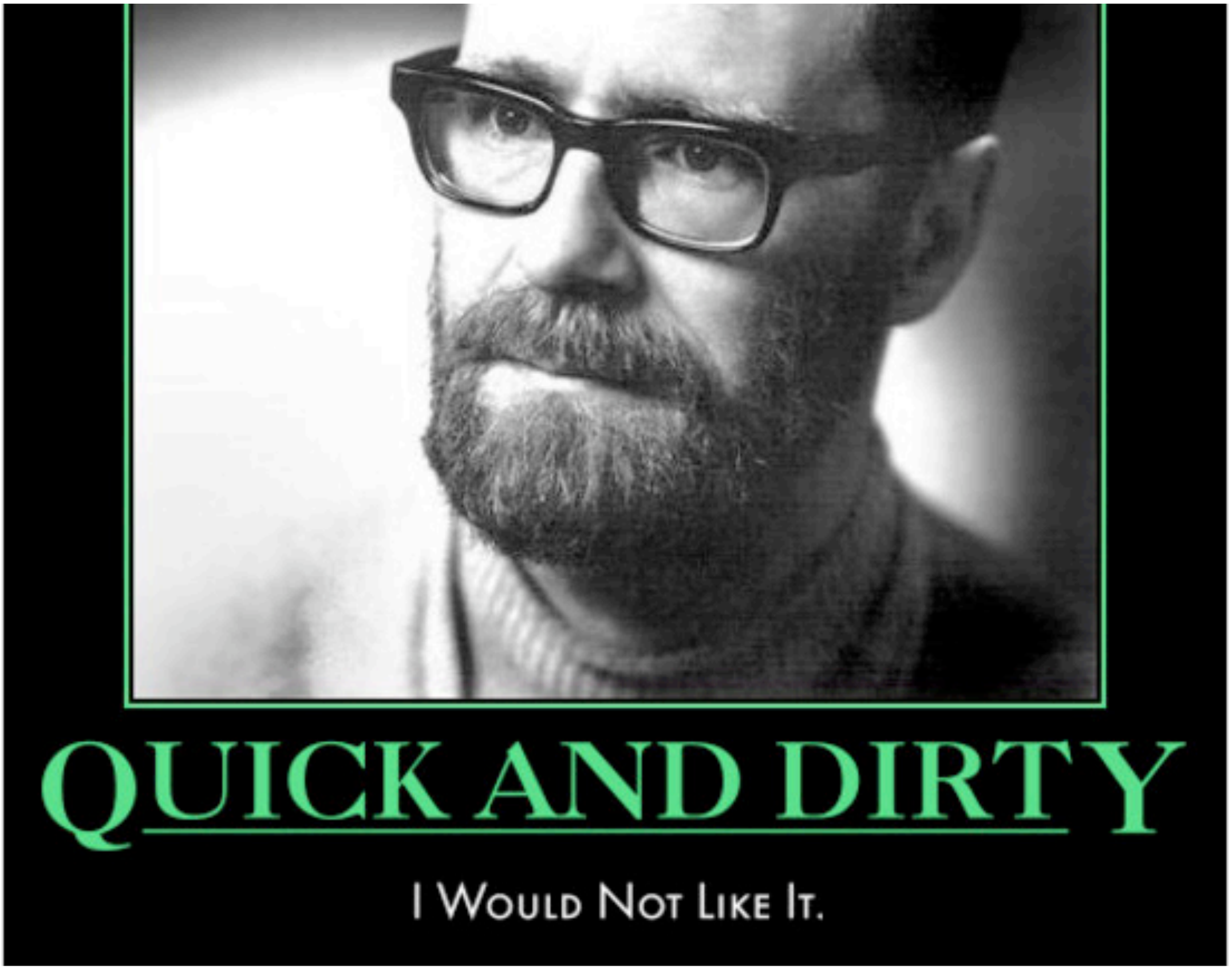

And now, a topic that I think combines both the geekiest, most beautiful aspects of coding with the cold reality of business: refactoring.


Refactoring is when you take “organised chaos” like this…


… and turn it into that beautiful unicorn.

While refactoring gives you some of the nicest warm'n'fuzzies you can get when coding, it's also a hard business decision to make, because, just like breaking up, it's never a good time to refactor. There are always important bugs to fix and important features to implement, and refactoring means that you're definitely not doing the latter. It's a trade-off: sacrifice time now so that you will (hopefully) get a payoff in the future because you can add features and fix bugs faster in the future.

Here's the main question: can you, as the lead coder, justify the refactoring to your product manager? Everybody has a tolerance threshold for code where things become so convoluted that you simply can't work productively. In the end, you, and not your product manager, are the best person to make the decision to refactor or not. Just be honest with yourself and make sure you can justify it, and don't do it too often.




Given that refactoring is a costly operation in business terms, get it right the first time. Everybody, especially me, loves rewriting stuff so it's more elegant, but chances are pretty good that once your first pass of the code is checked in, it's going to be in your code repository for a long, long time, and no amount of FIXMEs or TODOs comments are going to make you fix things later (because there's more important stuff to do). Respect your forefathers of Dijkstra and Knuth: make it right on try #1, and make it clean.

# Things You Should Never Do, Part I

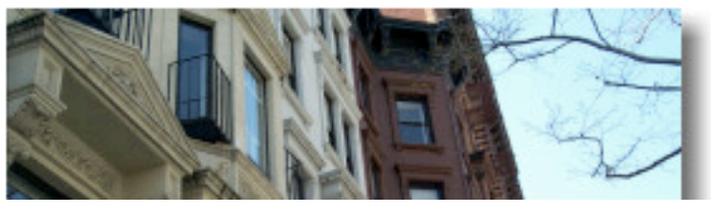
By Joel Spolsky
Thursday, April 06, 2000

Netscape 6.0 is finally going into its first public beta. There never was a version 5.0. The last major release, version 4.0, was released almost three years ago. Three years is an *awfully* long time in the Internet world. During this time, Netscape sat by, helplessly, as their market share plummeted.

It's a bit smarmy of me to criticize them for waiting so long between releases. They didn't do it *on purpose*, now, did they?

Well, yes. They did. They did it by making the **single worst strategic mistake** that any software company can make:

They decided to rewrite the code from scratch.



So, would you have rewritten Netscape?

Sure, Firefox is a ton better than Netscape ever was, and, with hindsight, rewriting it doesn't seem like a bad idea. However, three years is a stupendously long time to produce nothing. When you rewrite, your business is frozen in time. While your competition is moving, you are standing still. From an engineering perspective, refactoring is almost always always better than rewriting. Even if you're not advancing your product, at least you have software that's more-or-less functional throughout the refactor, and you often end up with a better product. (What do you think leads to better code: 1000 patches of 5 lines each, or a single patch of 5000 lines?)

OK to rewrite

Nevertheless, sometimes you can justify a rewrite, but if you do, you better have Iron Man on your team (or Jeff Lim, who's the guy on the right): you need someone who truly understands the entire system, and you better make sure that you can get the rewrite done relatively fast. I'd recommend rewriting parts of your application rather than re-do the entire thing from scratch.

True story: cineSync Mac 1.0.1 was a complete re-write of 1.0. 1.0 had completely separate code bases for the Windows and Mac versions, and we had intended to unify the two at some point. The Windows code was crufty enough that Jeff felt he could nail all the features we wanted for 1.0.1 almost as fast by rewriting the whole thing, and planned to make the code cross-platform in the process, so that the rewrite could be used as the basis for the Mac version. So, we worked for two weeks intensely, and sure enough, at the end of it, we had a cross-platform code base, so cineSync 1.0.1 for the Mac was a completely different beast internally even though it held the same look'n'feel. The business case here was that we needed a cross-platform code base in the long run, but we realised it early enough that we could get the job done in two weeks before the code got more complex. If we already had a sizable code base, the rewrite would have been far more risky, and refactoring was probably more appealing.


# Consider the Business Impact

The key point in all this is to ask yourself “what’s the business impact of what I’m doing right now?” whenever you’re coding. How much of a support burden is that shiny new feature going to be? Is this feature going to sell more copies? If so, do we need that revenue right now, or can we push it back a little and ride on the finances we’ve got and perform bug fixes instead, which will ensure a higher-quality product and lead to happier users?

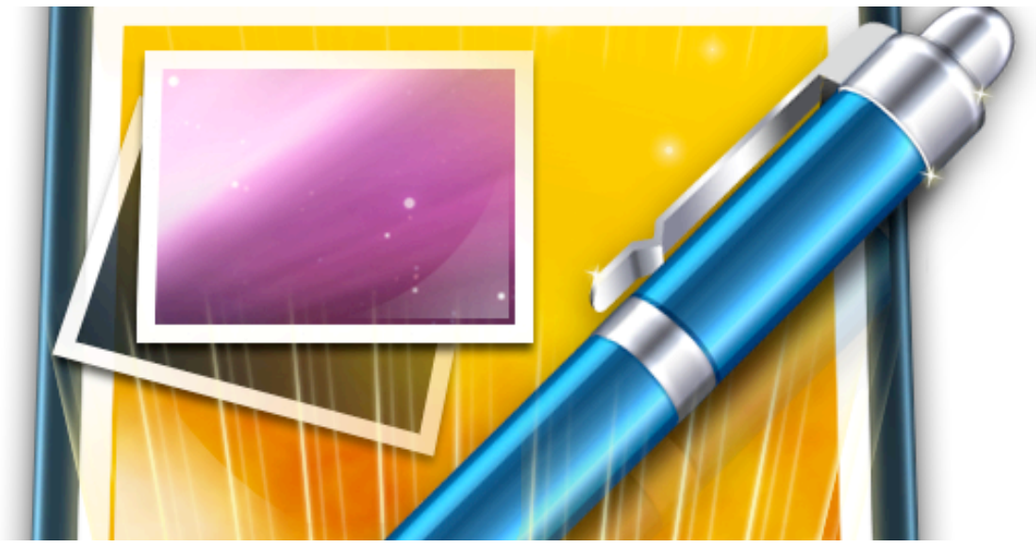
# Code Is Your Domain

"… developers (and especially development managers) tend to be pushovers. I believe this to be so to a large extent because they don't have a high enough regard for the principles of their craft. If you are a developer working with an overbearing "business" person, it's your responsibility to stand up for the system and make the case for the consequences of bad decisions (past and present)."

— Patrick Logan

*http://patricklogan.blogspot.com/2007/05/changes.html*

The product manager is ultimately the person responsible for all decisions, but in the end, you are the one who has the best judgement about the code. If you feel you need to do a refactor right now, do it. If you've been putting off including that crash reporter or auto-updater in your application, you need to justify that and be aware that your support guys are going to take a hit. If you really want to open-source that really cool part of your app, do it, but understand the trade-off: you might gain some fans with the developer community, but your product's standing still in the meantime. Understand that are you part of a larger team, and that the world does not revolve around you, but that you are also an essential, irreplaceable part of the team. Patrick Logan's quote sums it up beautifully.



## Thanks!

All images © by their respective holders, and were found via Google Image Search.


## Contact

http://algorithm.com.au/
ozone@algorithm.com.au

cineSync
share your vision

## Twitter

http://twitter.com/AndrePang