The Pragmatic Bookshelf

PragPub

The First Iteration

IN THIS ISSUE

* Writing an iPhone App
* iPhone, Meet Cucumber
* Pragmatic Life
* Managing Your Life Projects

Issue #2
August 2009

Contents

FEATURES

Writing an iPhone App 18
by Chris Adamson

Developing iPhone apps is in some ways remarkably like writing desktop apps—and in many ways jarringly different.

iPhone, Meet Cucumber 25
by Ian Dees

It's the hot development platform, and anything that speeds the process of getting your iPhone app finished and tested is welcome news. So Ian Dees shows how to drive an iPhone GUI from a Ruby test script.

Pragmatic Life 30
interviewed by Michael Swaine

A chat with Andy Hunt about the Pragmatic Life book series and about life in general.

Managing Your Life Projects 36
by Johanna Rothman

How to be more effective in your daily life by applying the principles of managing your project portfolio.

DEPARTMENTS

Up Front 1
by Michael Swaine

Your feedback on our first issue, our take on iPhone development, and more.

Choice Bits 3

Tweets, posts, and mermaid's tears from the interliving coral shoals of thought.

How Do We...? 5
by Dave Thomas

People often ask how we do what we do. This series explains…

Get a Life 7
by Daniel Steinberg

Fire up the backyard grill and roast a batch of beans.

Swaine's World 14
by Michael Swaine

Michael heads into the woods, reflecting on bold retreats and loving attacks.

Shady Illuminations 40
by John Shade

John Shade wonders whether you have to be social to write social software, and if so, whether that's a deal-killer.

The Quiz 45

A monthly diversion at least peripherally related to programming.

Calendar 47

Author sightings and other notable events.

Published monthly in PDF, mobi, and epub formats by The Pragmatic Programmers, LLC, Dallas, TX, and Raleigh, NC. E-Mail support@pragprog.com, phone +1-800-699-7764. The editor is Michael Swaine (michael@pragprog.com). Visit us at http://pragprog.com for the lowdown on our books, screencasts, training, forums, and more.

ISSN: 1948-3562

Up Front

What You Told Us

by Michael Swaine

Our first issue was one data point, and now here's another. Since two points determine a straight line, what do you think about our trajectory?

Welcome to the second issue of *PragPub*.

We're calling these first few issues the First Iteration. As firm believers in the power of agile development, Dave Thomas and Andy Hunt built an agile publishing company, and *PragPub* is intended to be an agile magazine. Where we take it will in large part depend on your feedback. Last issue we invited you to tell us what you want to see in a twenty-first century publication for software developers.

And you have.

You said (some of you said) you wanted to be able to have *PragPub* placed directly onto your bookshelf every month. You said you'd like to subscribe via email, or in Stanza. You asked to be able to subscribe to an RSS feed, possibly one for each story. You said you'd like to subscribe to our events calendar via CalDAV.

You were enthusiastic about our publishing simultaneously in epub, mobi, and pdf formats, but you wondered if we were going to do an HTML version. Some of you had trouble downloading the epub version. You asked for a clickable table of contents in the Kindle version. You found some editing errors (my bad). You suggested color and formatting changes to improve the look of the publication when printed.

You said you enjoyed the articles on Clojure, Andy Lester's article on keeping your job, the department in which we explain how we do some of the things we do, the quiz, and the John Shade column. You shared some ideas about what you'd like to see in future issues.

One reader wrote, "I like the magazine simply because it encouraged me to come here and get involved in the pragmatic forums. That's great because your books are excellent and it's great to read you want to start a dialogue with your readers. It also gives a feel for the personalities behind the names." Yep, that's what we had in mind.

So here's what we're going to do:

Color, formatting, and downloading issues: all addressed, we hope. We don't plan to do an HTML version now, since we're interested in going a different direction from surfing the web and reading a bunch of blogs; at least that's the plan for this First Iteration. We're looking into the subscribing possibilities: we'd like to make it as easy as possible for you to get *PragPub*. And we agree that the calendar should be a living document, so we're mulling that over.

As for content, in this issue you'll see, as you requested, articles on programming, as Chris Adamson shows how developing apps for iPhone

requires thinking different(ly), Ian Dees explores the possibilities of using Cucumber in developing apps for the hottest platform going, and Dave gives you a peek behind the Pragmatic curtain in another "how we do some of the things we do" department. We also shine light on other facets of the programmer's life in an interview with Andy, Johanna's advice on personal productivity, and Daniel's tutorial on how to become a coffee guru. Plus another quiz. And more John Shade. And other stuff. Anything else we can do for you?

Seriously. It's not a rhetorical question. We're eager to hear how we can make *PragPub* the kind of publication you want to read [U1].

-Michael Swaine, editor

External resources referenced in this article:

[U1] mailto:michael@pragprog.com?subject=PragPub

Choice Bits

Overheard on the Intertubes

Wading into the chatterstream in hip boots, we harvest the mermaid's tears from the internets.

What's Hot

Top-Ten lists are passé—ours goes to 11. These are the top titles that folks are interested in currently, along with their rank from last month. This is based solely on direct sales from our online store.

1	NEW	Metaprogramming Ruby
2	1	iPhone SDK Development
3	3	The RSpec Book
4	NEW	Security on Rails
5	4	Agile Web Development with Rails, Third Edition
6	6	Programming Ruby 1.9
7	7	Programming Clojure
8	2	Language Design Patterns
9	NEW	Programming Scala
10	8	Writing Your First iPhone Application
11	10	Core Data

And Here's What Folks Are Saying…

- *Maybe the thoughts we generate today and flick around from mind to mind… are the primitive precursors of more complicated, polymerized structures that will come later, analogous to the prokaryotic cells that drifted through shallow pools in the early days of biological evolution. Later, when the time is right, there may be fusion and symbiosis among the bits, and then we will see eukaryotic thought, metazoans of thought, huge interliving coral shoals of thought. The mechanism is there….* — Lewis Thomas, *The Lives of a Cell*, 1974.
- *Want to Find Marijuana? There's an App for that.* — Hunt Henning, student
- *The App market will grow up & be as big as the internet by 2020… although that's 5 internet lifetimes!* — Matt Yorke, Englishman abroad
- *Well played, Pragmatic Programmers, well played. The inaugural issue of PragPub, an electronic magazine for programmers… has a cameo appearance for LOLCODE.* — lolcode.com
- *I've only just got a Wave account and I've already been rick-rolled by a RickRoll Wave bot.* — John Resig, creator of JQuery
- *OK, I'm over it. Java is just really not all that much fun.* — Carl Schuyler, programmer
- *Laundry tip: Drying jeans on "ex. low delicate" = not effective.* — Andy Lester, Working Geek
- *My attitude, as a user, is that if a site is going to track me, I want them to do it openly, using cookies. Cookies offer me less transparency and control than I would like, but the alternatives are worse. If I were writing a self-regulation code for the industry, I would have the code*

require that cookies be the only means used to track users across sites. — Ed Felten, Freedom to Tinker

- *Amazon.com reviews from Jeffrey P. Bezos. Includes milk, cookies, cheese snacks, binoculars, and a Cory Doctorow novel. Unfortunately, the cookie items reviewed are no longer available so we're not able to share in "snickerdoodles [that] were the best I've ever had."* — Matt Mullenweg, creator of WordPress
- *Sure I sold you robot insurance. But you were attacked by a cyborg. Not covered.* — KAgroX, Congress watcher
- *It's big industry news, but how are non-Yahoo end-users affected by the Microsoft-Yahoo search deal? That's what I'm thinking about.* — Mitch Kapor, co-founder of Electronic Frontier Foundation
- *A world where people regularly use 9 year old operating systems is not a healthy computing ecosystem.* — Jeff Atwood, Coding Horror
- *NSString's* `caseInsensitiveCompare:` *should not be confused with* `caseinSensitiveCompare:`*, which is for lactose intolerant people. -Cocoa tip of the day* — Andy Lee, creator of AppKiDo
- *Extreme Programming excels at the ordinary and allows the extraordinary. @KentBeck asks, and I answer: no need to promise paradigm shifts.* — Ward Cunningham, creator of the Wiki
- *Going thru old photos… Here's a pic of me & Ray Manzarek after the "Craigslist" session.* — "Weird Al" Yankovic
- *Saw a sign in CO Springs restaurant: Free Smells. I think I've seen that sign on many instances of code.* — Gerald Weinberg, Software Psychologist
- *Chords for iTunes is like Guitar Hero for grownups.* — Bob LeVitus, Dr. Mac
- *Hoping the Beatles will release their albums at the same time as 10-inch iPod touch.* — Tony Bove, Haight historian and iPod developer
- *Policies are organizational scar tissue. They are codified overreactions to unlikely-to-happen-again situations.* — Jason Fried, founder of 37Signals
- *My autobiography will be titled "Sociopaths, Borderlines, and Autistics," subtitled: "How I cope with dealing with humans."* — Bram Cohen, creator of BitTorrent
- *A TV show that shows you wacky Internet videos is like a restaurant that serves you Dominos Pizza.* — Andy Ihnatko, friend of Roger Ebert
- *If you had asked me which comic strip was going to quote Pynchon this morning, I would not have come up with Sally Forth.* — Chris Espinosa, Apple employee number 5
- *Only in this century have we been brought close enough to each other, in great numbers, to begin the fusion… and from now on the process may move very rapidly…. What we need is more crowding, more unrestrained and obsessive communication, more open channels, even more noise, and a bit more luck…. We are simultaneously participants and bystanders…. As participants, we have no choice in the matter; this is what we do as a species. As bystanders, stand back and give it room is my advice.* — Lewis Thomas, *The Lives of a Cell*, 1974.
- *Just participated in a group sing along on a sidewalk in NYC. This must be what being a Bollywood character feels like.* — Chad Fowler, a Passionate Programmer
- *When I start writing something long, I know exactly which direction I'm heading, but have no idea where I'm going.* — John Gruber, Daring Fireball

How Do We...?

How Gerbils Make Sausage

by Dave Thomas

People often ask how we do what we do. This series explains…

Just as with our books, these magazines are written in plain text, using PML, our XML-based markup language. And, just like the books, we collaborate by checking the various components in and out of our version control system.

We use Jim Weirich's Rake utility to coordinate all the tasks associated with building and distributing the content. Typing `rake magazine.pdf`, for example, builds the PDF version—you can build epub and mobi versions too. Once we're happy with an issue, `rake upload` creates all three versions and uploads them to S3 for your reading pleasure. Other Rake tasks do things like generate cover images for the online site, create a Textile format table of contents for the online description, and so on.

Our PML markup is simple but flexible—common stuff is easy and relatively terse. Here's the start of the body of this article:

```
<body>
  <the-gerbil-asks>
    How do you create this magazine?
  </the-gerbil-asks>
  <p>
    Just as with our books, these magazines are written in plain
    text, using PML, our XML-based markup language. And, just like
    the books, we collaborate by checking the various
    components in and out of our version control system.
  </p>
```

We can extend the markup to add custom tags for any issue (in the same way we can extend it for each book). In this issue, for example, we needed to draw a Sudoku diagram. We added markup that let us do:

```
<sudoku>
  <s-row>.A.FK....</s-row>
  <s-row>.K.J..A..</s-row>
  . . .
  <s-row>.J.I...R.</s-row>
  <s-row>K.R.....F</s-row>
</sudoku>
```

From PML to PDF (and the rest…)

The first step in creating a readable document is to deal with any source code in program listings. A preprocessor (written in Ruby) scans the PML files, combining them into a single XML document. Along the way, it looks for our special code inclusion tags, finds the appropriate source files, syntax highlights them, and inserts the result as valid XML into the combined document.

Once we have this combined document, we create PDF, mobi, and epub files from it. Let's look at PDF first.

For the print books, we currently use XSLT to take this document and convert it to TeX. On the TeX side, we use a combination of the MEMOIR package and a whole bunch of our own style macros to create the final product. For the magazines we do it differently—we're experimenting with using xsl:fo. A XSLT transform takes our XML and converts it to flow objects, which then gets rendered into PDF. Right now we're using RenderX[U1], which does a nice job of PDF generation.

For mobi files (used by the Kindle), we use a different XSLT transform to convert our XML into a very simple (and very nonstandard) HTML. This includes some special tags and id attributes that tell the Kindle reader things like the location of the table of contents. This HTML then goes through the html2mobi utility, part of the MobiPerl package. (We're also looking at using Calibre in future). One of the joys of the Kindle is that the internal markup doesn't seem to be published anywhere, so there's a whole bunch of experimentation and guessing involved.

For epubs, we use a third XSLT transform to create an IPDF compliant directory tree containing the various OPF files and content. However, we've noticed that feeding epub readers a document built using a single HTML document bogs them down terribly, so before we package the content, we run a utility that uses the Ruby NokoGiri library to split the master document into multiple sections, rebuilding the various resource indexes to reflect this new structure. (We originally did this splitting in the XSLT, the way the DocBook transforms do, but this is horrendous as the splits are not hierarchical. If you want to see some unbelievably complex XSLT, look at the way DocBook creates epubs.)

In the middle of these steps, we also use a Ruby program to convert the image files to a format suitable for the target readers.

Of course, this is what we did today. Tomorrow, it'll probably be different.

About the Author

Dave Thomas is one of the Pragmatic Programmers.

External resources referenced in this article:

[U1] http://renderx.com

Get a Life

Roast Your Own Beans: a Guide for the Dedicated Caffeinophile

by Daniel Steinberg

There's more to life than coding—such as the search for the perfect cup of coffee.

I can't imagine coding without coffee.

It could come from my background in mathematics. After all, mathematician Paul Erdos famously said, "A mathematician is a device for turning coffee into theorems."

Even if all mathematicians are coffee drinkers (which they aren't), programmers definitely come in different flavors when it comes to the beverage we choose to give us that (no plug intended) jolt. Some prefer tea, some caffeinated soft drinks, but a whole bunch of us use coffee to start, restart, and sustain us through the day.

But if we drink coffee only to get that boost, we are missing out on one of the subtle, simple pleasures of life. A great cup of coffee is complex, in some ways more complex than a fine wine. Yet most of us drink coffee without thinking much about the taste. I think that's because most of the coffee we drink is pretty bad.

I'm going to help you to make great coffee. Specifically, I'm going to show you how to take advantage of the summer grill season to improve the freshness and flavor of your coffee. You won't want to roast your own coffee beans on the grill all the time—but it's a fun thing to try, and it can definitely improve your coffee-drinking experience.

Why we roast

You can make huge improvements to your coffee by insisting on freshness. You already know that you want to brew the coffee right before you drink it, because you've had nasty coffee that's been sitting on a burner for hours. It's pretty easy to make this change at home. Pick your favorite method of brewing and make only as much as you want right now.

The next step in freshness is to grind your beans just before you brew. Buy yourself a decent burr grinder (not a cheap blade grinder, which will mangle the beans) and you'll never have to buy your beans pre-ground again. As soon as coffee is ground, oxygen starts to attack it and rob it of its flavor. Buying whole bean will extend the life of the coffee you bring home.

The third step is to get freshly-roasted beans. In his book *Home Coffee Roasting* Kenneth Davids says that the moment coffee "is roasted it begins a rapid, relentless journey from flavorful to flavorless." Davids says the biggest villains in this story are moisture, heat, and oxygen. He explains that the physical structure of the bean and the carbon dioxide produced by roasting initially protect these flavors. As time passes, carbon dioxide starts to leave the roasted beans, and this allows the oxygen to start attacking the flavors.

So if the beans were roasted just before you ground them and brewed your coffee, you would be doing all that you possibly could to preserve the beans' physical structure and release the promise of the bean in your cup.

OK, you can grind the beans at home and you can brew the coffee at home, so why not home-roast the beans as well?

Relax; I'm not going all Martha Stewart on you. You don't need to plant coffee beans and harvest the cherries and then dry them. And you don't need to buy a coffee roaster to start roasting your own. If you have a grill or a hibachi, we'll have you roasting in no time.

Be careful

Just like any other hobby, roasting is a slippery slope. You'll start out using this low-cost grill approach and the next thing you know you'll be buying expensive equipment, unable to stop yourself.

I roasted my first batch of coffee about ten years ago. I followed the instructions and carefully roasted half a pound in my oven and set off all four smoke detectors in our house. That was the year that Kim encouraged me to buy my first home roaster and to use it outside.

I started with a simple fluid bed roaster. It was glass on the outside so I could watch the coffee beans turn from light green to tan to chocolate brown. I couldn't roast very much at a time and the roaster only lasted a couple of years.

I moved up to a drum roaster. I could no longer see inside but I could smell the different stages of the coffee as it went from a grassy smell to a toasted grain smell to a coffee smell. And if I listened carefully enough I could hear the beans cracking, which allowed me to stop the roast when it had gotten to just where I wanted it.

Coffee beans are kind of like popcorn. As they cook, they crack. Except that coffee beans crack twice. After the first crack the coffee produced is quite good. But if you like a darker roast like an Italian, a French, or an espresso, then you want to let the coffee go well into the second crack. The beans will start to get oily and the flavor of the bean will start to be overpowered by the flavor of the roast. Have you ever noticed that when you get a French roast they don't tell you French roasted what? It's because you aren't tasting much of the bean any more. You're tasting the roast. Some chains thrive on these dark roasted—burnt beans.

Good, freshly roasted coffee makes such a difference—what? Oh, you're sold, and you want me to tell you how to roast your own coffee beans? OK.

Home roasting the easy way

It turns out, if you have an outdoor grill, nothing could be simpler. I've used a small charcoal grill that I bought for under $40. I've also used a pan, a wooden spatula for stirring, a dish towel to hold onto the hot pan, and a strainer for cooling the coffee down afterwards. You can get started with little to no investment.

You'll need a source of green coffee beans, to start. I buy my green beans from Sweet Maria's online but I used to buy from local coffee shops. They generally

would charge me half as much for green beans as they did for roasted. See, I'm actually *saving* you money. You might want to tell your significant other.

Fire up the grill and let the pan come up to temperature. You don't need a special toy for this but I have a radar thermometer. I tried temperatures from 300 degrees to 700 degrees and found that about 400-450 worked best for me.

Fill the bottom of the pan with green beans and immediately stir; within a minute the beans will start to color as shown on the right:

You can see that the beans give off a lot of smoke—you don't want to do this in the house.

After the first crack, a paper shell comes off of the bean.

I give the pan a light blow to separate this chaff but that can wait until the cooling step. Be careful! Don't blow the embers around, or burn yourself in any of the other ways that you will discover.

In the roasting illustrated here, I took different batches to different degrees of doneness. I like doing this when roasting. You get different flavors from the beans at different degrees of roast. Instead of blending different beans together,

as your favorite roastery might do, I'll often blend different roasts of the same bean together. Here you can see a wide range.

When the coffee bean is ready, pull it off of the coals and dump it into a strainer. Shake the beans around to cool them off. You do this to stop the cooking and bring the temperature down quickly.

Some people mist their coffee with water at this point, but if you don't know what you are doing you can do more harm than good so I would just woosh the beans around until they have cooled enough that you can touch them.

The Payoff

Now go brew a pot. You can let the beans rest. Some beans benefit from resting half a day or more, but there's no harm in grinding up your beans and brewing a fresh pot right away. And there's much satisfaction in tasting the results of your efforts.

The final step in enjoying a great cup of coffee is conscious tasting. With a sip of coffee in your mouth breathe in through your mouth like you are tasting wine. Engage the sides and back of your tongue. You should discover a wide range of flavors you've never noticed before.

About the Author

Daniel is the editor for the new series of Mac Developer titles for the Pragmatic Programmers. He writes feature articles for Apple's ADC web site and is a regular contributor to Mac Devcenter. He has presented at Apple's Worldwide Developer Conference, MacWorld, MacHack, and other Mac developer conferences. Daniel has produced podcasts for Apple featuring the work of developers and scientists working on the platform. He has coauthored books on Apple's Bonjour technology as well as on Java Programming and using Extreme Programming in Software Engineering classes.

Swaine's World

Happy Campers

by Michael Swaine

I have another life.

My partner Nancy and I live on, and Nancy manages, a small organic farm in Southern Oregon, plus an on-farm restaurant and bakery that use our farm's produce. Although Nancy runs the place I help out and I interact daily with our twenty-some (and mostly twenty-something) employees.

Every year about this time we take all our employees and their families out into the woods to a pretty spot next to a mountain lake for a three-day camping retreat. We set up a fairly elaborate outdoor kitchen, and with three professional chefs on staff, the meals are definitely upscale campfare. Everyone who plays a musical instrument brings it. People hang out together around the campfire at night and during the day organize into ad hoc groups to hike, climb the nearest mountain, bike, swim, or row on the lake.

The retreat is one of the smartest things we do for our business. On the surface it's a nice way to acknowledge our staff for their hard work, but the real stealth benefits come from their interaction over the three days. They get to see their co-workers as three-dimensional human beings with families and knobby knees and good or not-so-good-but-who-cares voices. They work together voluntarily in self-organizing teams to gather kindling and build fires, clean the grill, cook, clean up, or hike to the store for ice. They are no longer three separate staffs—restaurant, farm, bakery—but one group of people solving problems together and discovering what they have in common.

Putting together this annual retreat, collecting and transporting all the food and supplies, setting up and breaking down camp for perhaps fifty people, is exhausting, but we wouldn't think of skipping it. It's a great experience for all concerned.

Imagine, though, if we were actually getting productive work done as well as having this bonding experience. Imagine if we were also honing our job skills. I can assure you that no job-skill honing is going on at our staff retreats. But there are events where you can experience both bonding and professional growth. When you find one of these events, it's magic.

Just the Good Parts

In a posting on *Dr. Dobb's* CodeTalk blog, Nick Plante contrasted coding camps with traditional conferences. We've all been to a conference where every session just served up information we could have downloaded and read or watched at home. "The real reason I go," Nick said, "and I think the real reason most people go—is for the hallway discussions, late night hack sessions, and extracurriculars."

From this universal frustration with traditional computer conferences have sprung unconferences (barcamps, podcamps) that try to encapsulate just the good stuff. There's a definite spirit of The Simplest Thing that Could Possibly Work in these minimalist events.

Nick asks what if we combined an unconference with a retreat (I'm paraphrasing), and he notes that it's been done, is being done, at Rails Camp. 30 to 50 people in a large house in a remote location for a weekend, living together, eating together, working together. No internet. Much playtime. Much sharing of ideas and expertise. Presentations, if any, are ad hoc. Cost minimal.

"Camps," Nick told me later when we chatted about this, "seem to force everyone who attends to be an active participant, and share their knowledge and interests with others while working and playing together (the playing part seems as important as the work, if not more so)." He considers such camps to be unique among tech events in the intimacy and learning opportunities they provide.

Washing Dishes with Woz

What came to my mind immediately when reading Nick's post was an event I attended in (gulp) 1984.

I had the privilege back then of being invited to the original Hackers' Conference, a true computer camp before the term was coined. Inspired by Steven Levy's book *Hackers*, organized by Stewart Brand and the Point Foundation, it was, if you go by Ted Nelson's description, "the Woodstock of the computer elite."

Elite? I certainly thought so. Richard Stallman, Richard Greenblat, Steve Wozniak, Ted Nelson, John "Captain Crunch" Draper, Bill Atkinson, Lee Felsenstein, shareware pioneers Bob Wallace and Andrew Fluegelman, and over 100 more legendary hackers—in the best sense of the word. We gathered at remote Fort Cronkhite in the Marin Headlands where the wind was wild and the electricity iffy. (Yes, we had electricity back then. Just stop it.) We lived and ate and talked software and software politics for 48 straight hours. I remember washing dishes next to Woz, and it echoed my own thoughts when, in the opening circle of introductions, Brand said, "I'm Stewart Brand and I'm just here for the contact high."

So what did we accomplish? I don't know, but I know it was magic. I know we talked about the future of the Hacker Ethic. It was at this conference that Brand famously said, "information wants to be free." Nobody remembers that he also said that information wants to get paid for. We didn't resolve that conflict, but we recognized it a quarter century ago. And we accomplished this, ineffable though it may be: everyone who attended that conference was enlarged by the discussions we had, and we took the insights we gained with us into our subsequent work and lives.

When you get the right people together in the right setting, magic can happen. I imagine Dave and Andy have some of the same feelings about the Snowbird Conference where the Agile Manifesto was written.

Intimate Attacks

I'm a writer, so the workshops that have most affected me over the years have mainly been writers' workshops. There's a model for the most effective of these, called the Clarion model. Nick spoke of the intimacy of software camps, and a Clarion workshop is as intimate as open-heart surgery.

In a Clarion workshop, victims, I mean participants, usually sit or lounge in a rough circle, where each in turn has to sit silently while every other participant attacks, I mean critiques, the victim's baby, I mean manuscript. The process is brutal at the start, but because everyone is both attacker and attacked, most participants soon learn valuable lessons in how to take and give criticism. And until you've learned to take criticism you will always meet barriers to growth in your craft.

There's a leader, typically a Famous Writer (I've Clarioned under Roger Zelazny and Ursula LeGuin, among others), and although their role is crucial, it's extremely subtle. Often the leader appears to be there mainly to prevent fist fights. All morning you workshop, all afternoon and much of the evening you write, and in the gaps there is socializing, which tends to be as intense as the workshopping. At the end of a week or six weeks of this you feel completely wrung out and insanely exhilarated.

I don't want you to think that the blood-letting of this process doesn't hurt. It hurts like anything when one clueless drone after another completely misses the charm of the precocious prose you gave birth to at four that morning in a blaze of tequila-inspired brilliance. And it hurts in a different way when you understand that if nobody gets it, maybe it isn't really there. But it's also true, as Richard Gabriel says, that the criticism from your fellow participants is a gift, given lovingly, even if neither giver nor receiver sees it that way at the time, in reciprocation for your astonishingly generous gift of your work in progress.

And years later, when you look back on these intense experiences, that's exactly how you see it.

I think that any programmer could probably benefit from a writers' workshop. Not the old how-to-sell-you-novel kind, though. It would have to be a new kind of writers' workshop, one that shows you how to find the elephant in the block of marble. You know the old joke: "How do you carve an elephant? Just get a huge block of marble and chip away everything that doesn't look like an elephant." It's only a joke on one level: there really is an elephant in there to be discovered. To the sculptor who works in driftwood, it's not a joke at all. Writing is more like working in driftwood than working in marble.

Do software developers sculpt in driftwood or marble? In either case, writers' workshops are not that different from software camps, and nobody makes that point more clearly than Gabriel in his book *Writers' Workshops & the Work of Making Things*.

"In an open-source project," Gabriel says, "the gift of source code is reciprocated by suggestions, bug reports, debugging, hard work, praise, and more source code." But this gifting takes courage. In a writers' workshop or in an open-source project, "[t]he author gives the gift of a work in progress to a group of writers each just as afraid as the other, and that group will own the work for a while

and give back the gift of suggestions. By carefully crafting the setting and ritual of how those authors work together, the gift exchange can be made to work beautifully...."

That's why it's so important to check out gatherings like Rails Camp. When these things work, it's magic. If you have the opportunity to experience that magic, grab it. It will change your life.

About the Author

Michael Swaine is the editor of *PragPub*.

Writing an iPhone App

Thinking Different(ly) about the iPhone SDK

by Chris Adamson

Now that we've had a year to get the 99-cent "ring-tone" apps out of our system, are you ready to write some serious software for the hottest platform going?

It's been a few years since Apple has used its television ads to tell us to "Think Different." But maybe their actions speak louder than words: with the iPhone SDK, they've brought a desktop-like platform and toolset to mobile developers, while the App Store offers a hitherto unheard-of number and variety of applications to users. The competition sure isn't thinking different: within a year of the App Store's launch, "me too" app stores are popping up on other platforms almost monthly.

But what about you? Is this a field of programming you should get into? In this article, we'll take a brief look at what's involved in iPhone development, focusing on language, frameworks, tools, and distribution. If you like new challenges, I think you may well be intrigued by what this new platform has to offer.

Code Different: Objective-C

Chances are you've heard that iPhone applications are written in Objective-C. Recruiters certainly have: online iPhone job postings seek candidates with an arbitrary number of years of experience in this ostensibly exotic language. Yet Objective-C is fundamentally like other object-oriented languages such as Java and C++, and a developer can learn its core concepts in an afternoon.

Objective-C is a "strict superset" of C, meaning only that any valid C program compiles in an Objective-C compiler. Primarily, Objective-C adds a Smalltalk-inspired object layer atop C. In the abstract, the concepts aren't unlike other OO languages: classes use single-inheritance (like Java and unlike C++), protocols allow you to define methods that can be implemented in multiple classes (similar to Java interfaces, but with the option of marking some methods as optional), and you call methods on these objects.

Well, you don't *really* call methods, *per se*. While it is convenient to use that terminology, Objective-C technically sends messages. This is an important distinction, as it allows for interesting flexibility in the language. What we see as a method call is maintained by the Objective-C runtime as a string naming what is to be called. In effect, the runtime asks the receiving object if it implements such a method, and if so, the runtime sends it the parameters. The practical upshot is that for a compiled, strongly-typed language, Objective-C allows for dynamic traits we typically associate with scripting languages.

Oh, and the square braces. And the really long method calls. We can't let this syntax escape our gaze, since the first glimpse of Objective-C code seems to give some developers shivers. Square braces denote the sending of a message, with the name of the message/method split up to denote its arguments, like the following:

```
[target message: argument with: anotherArgument];
```

In this pseudo-code, target is the object being operated on, and message:with: identifies the message to be sent (written with just the message name and not the argument types, it is called a selector). While seemingly verbose, the style is highly self-documenting. Other C-based languages may send you scurrying for the docs when you come across a function or method call with three or more comma-separated arguments, but a similar Objective-C call speaks for itself, like the following:

```
NSString *myShortString =
  [myBigString stringByReplacingOccurrencesOfString:@"foo"
    withString:@"bar" options: NULL range:myRange];
```

Notice in this example, the * character on our local variable. This is your constant reminder that despite the consistent and coherent world of objects that has been created for you, at the end of the day, you are still in the land of C, with every object an honest-to-goodness C pointer. For some developers coming over from scripting languages, the asterisk is a strange bit of arbitrary syntax, but those who've worked with C know well the power and danger of working with pointers. Free the pointer and then mistakenly re-use it, and you will likely be headed to a memory-related crash (often the dreaded EXC_BAD_ACCESS that so baffles newcomers). The realities of C can be ignored for a while, but in time, you should familiarize yourself with the wisdom of Messrs. Kernighan and Ritchie, or any modern equivalents you can find.

Call Different: Cocoa Touch

For many of us, what makes a platform is not the language but the libraries. After all, on the Mac, you can develop Mac applications not only in Objective-C, but also Ruby and Python. The iPhone remains Objective-C-only, but as in Mac development, you write your applications in the rich Cocoa environment. Technically, Cocoa is not a set of libraries, but frameworks, which in Apple's use are collections of code libraries, documentation, localizations, and any other resources (perhaps images and sounds) needed to use them.

These frameworks are much-tested and refined from years of use. Most of Cocoa traces back to the original NextStep operating system developed over 20 years ago, which evolved into Mac OS X and then formed the basis of the iPhone OS. While the UI layer is new for the iPhone, specifically designed for use on a mobile touch-screen device, core classes like strings, collections, dates, URLs, etc., are the same as on the Mac, well thought-out and well-understood.

Using Cocoa Touch, as it's called on iPhone OS, may require a shift in attitude. While other platforms are customized through subclassing—Java exhibits this to an extreme—Cocoa places a high value on making its classes usable as-is. In many cases, you customize a class through the delegation pattern. With this approach, you supply an object with a delegate that implements some methods of a protocol, providing your custom behavior in these callbacks. For example, every iPhone application should, at startup, set the delegate property of the UIApplication object that represents the application's existence and role within the iPhone OS. This delegate implements the UIApplicationDelegate protocol, and is called when events of interest to the application as a whole occur: the

application has started up, the application is running low of memory, the application has received a message from Apple's "Push Notification Service." To the developer, it looks and feels like a lot of callback or listener patterns, but to Cocoa, the idea is that a given class cannot know how it is to react to a specific situation, and relies on its delegate, if any, to act. In this case, the UIApplication object doesn't know what to do when memory gets tight, so it throws control to the code you've provided as the app delegate and says, "hey, do something!"

Delegation is one of the most common, most distinctive traits of Cocoa programming, one that is made possible by Objective-C message dispatch: delegates are set late, and the Objective-C runtime inspects delegates to see if they respond to a delegate message at all, moving on if they don't.

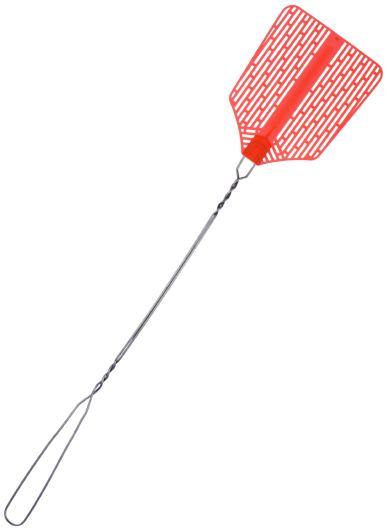
Delegation is one of the most important Cocoa patterns, but it's not the only one. Model-view-controller is present and accounted for throughout Cocoa Touch, most obviously in the GUI classes: onscreen views subclass UIView, and you'll typically manage them in subclasses of UIViewController (yes, you do *some* subclassing!).

A crucial pattern to learn is Cocoa's scheme of memory management. On the Mac, Objective-C offers garbage collection, but this is absent on the iPhone. Instead, you must employ a scheme of reference counting. All Cocoa objects maintain a count of references to themselves. Objects begin life with a reference count of 1, which is incremented any time another object wants to share in ownership of the object, by calling the retain method. Owners that have no further use for an object release it, decrementing the reference count. When the count goes to 0, the object is freed. It is challenging to get used to, particularly if you've had a garbage collector cleaning up after you, but the basic rules are simple: if you create an object (by means of the alloc method), or retain it, you must call a corresponding release at some point. Conversely, if you acquire an object by other means, such as a "getter"-style call, you don't own it and must not release it.

It's also worth noting that iPhone programming isn't all about Cocoa Touch. The platform includes a number of third-party libraries that are largely independent of Cocoa Touch, such as OpenGL for 2D and 3D graphics, OpenAL for spatialized sound, BSD sockets, and an API for using the included SQLite database. These are all programmed in procedural C, as are a handful of Apple frameworks, some of which offer a greater level of control (at the price of complexity) than their Objective-C counterparts. And in a few cases, such as graphics (Quartz) and audio (Core Audio), there may be no Objective-C counterpart, meaning you'll have to use procedural C directly. Fortunately, Apple's C-based frameworks generally employ design patterns similar to what's used in Cocoa, in some cases forging quasi-object structures that can be cast to Objective-C objects at no cost.

Build Different: The iPhone SDK Toolset

Thus far, I've discussed the language and frameworks in the abstract, without touching on the craft of code, where fingers meet keys, and code meets compiler. The SDK includes a compelling set of tools to master.



You'll develop your applications with Xcode, an Integrated Development Environment that will be familiar to users of other IDEs, and is more or less comparable to its counterparts on other platforms. When you begin an Xcode project, you choose the style of application you want to write (an app with a single view, one that flips between two views, one that navigates through many views, etc.), and it sets up the GUI resources and a basic set of application delegate and view controller classes to get you started.

Xcode sits atop the classic GNU C compiler and debugger, but makes their use far more appealing than they could ever be on the command line. As you set breakpoints in your code and step through statements, you can mouse over the variables in your source to see their current values and members pop-up as you mouse over them, or view them in a more traditional debugger window with source, assembly, and variables. Better yet, this works not only with the capable iPhone Simulator (which runs your iPhone application in a Mac process on your desktop), but also remotely with your code running on an actual iPhone or iPod Touch device.

The SDK also provides a set of performance tools to help you find and fix slow spots in your code. Shark lets you get up and running quickly with an overall assessment of which function and method calls consume the most of your app's running time. For a more detailed view, you can use Instruments, which samples your application as it runs and lets you view your app's runtime history on a timeline, examining specific points at which the app's CPU or memory use spiked, with the ability to trace down to specific calls. To top it off, the open-source Low-Level Virtual Machine (LLVM) group's Clang Static Analyzer project works with iPhone SDK projects, allowing you to perform a build-time analysis of your code to find errors like memory leaks, unreachable code paths, unused and uninitialized variables, and more.

Design Different: Interface Builder

iPhone applications are also distinguished by their appearance, something that is greatly facilitated by the most unique of the SDK tools, Interface Builder. Many platforms have GUI builders, but few are as tightly integrated as Interface Builder.

The most distinctive feature of Interface Builder is that unlike so many other GUI builders, it does not generate code that you then compile. IB creates honest-to-goodness Cocoa Touch objects, allows you to customize them, and then serializes the objects to disk inside nib files. At runtime, your application loads the nibs, deserializing the GUI objects and adding them to the GUI.

"But," you might ask, "without code, how do I provide the GUI components with any behavior of my own?" In one of the most unique (some would say peculiar, others would say awesome) patterns of iPhone and Mac development, you use IB to relate objects to variables and methods in your code. An outlet, marked with the IBOutlet keyword, is a variable that is meant to refer to an object loaded from a nib. You associate the outlet with an object by connecting dots in IB, an association that is stored in the nib and is "wired up" when the nib loads. Similarly, an action is a method that can be called from an object created in IB, commonly from an event generated by a GUI object. For example, your view controller code might have an IBAction that is connected

to a button's "Touch Up Inside" event (i.e., the user has tapped and released the button), which does some work and updates a table in the GUI, which it accesses by means of an IBOutlet to the table.

Some developers seem to over-think IB, and look for ways to retreat to the seeming safety of creating GUI objects in code. The key is to understand that the objects created in IB are honest-to-goodness objects, stored in the NIB along with enough information to bring them to life in your application: what kind of class "owns" them (and therefore can provide outlets and actions), the actions and outlets to connect to, etc.

So why do it this way? One advantage of IB is its tolerance for radical interface overhauls. If your GUI builder generates code, it's hard to avoid modifying that code, or at least depending on its implementation details. With IB, there's a great deal of decoupling of GUI and logic, connected only by the outlets and actions. Designers can own the GUI and make profound changes, in many cases acting independently of coders.

Deploy Different: The App Store

Coming from the desktop or webapp world, it's hard to appreciate how locked down mobile platforms have traditionally been. I often relate the anecdote about being at a mobility conference just a month before the unveiling of the iPhone SDK, attending a role-playing panel discussion in which a carrier representative openly mocked the romantic notion of the indie developer toiling away on a breakthrough application. She asked the panel member representing the mobile developer if he was interested in going "big time," with zero interest in his application if he wasn't.

And two years ago, that would have been it: many carriers and handset makers used security technologies to cripple or utterly disallow third-party applications, allowing users to at best see a handful of apps from big-name corporate partners. A forum post I saw years ago put it perfectly: the carriers couldn't stand the thought of you making a buck off their network if they don't keep 99 cents.

Instead of you being left with just a penny, how does 70 cents sound? Or, if you like, give your app away for free, and pay nothing for hosting or bandwidth. Those are the options offered by Apple's App Store, the primary means of delivering iPhone OS applications to iPhones and iPod Touches. You enter into an agreement with Apple, deliver your built-for-distribution application with some metadata (description, screenshots, support e-mail, etc.), and the App Store lets users download your app to their devices. As a developer, you don't have to deal with hosting, authentication, billing, or any of the other tasks that developers don't particularly like to deal with.

The openness of the App Store has been revolutionary, with indie developers offered as much access to the end-user as big companies enjoy, and with the indies often outshining their deep-pocketed peers. With a billion and a half downloads of the more than 60,000 apps available in the first year of the App Store, it's comical to see other platforms and carriers, the same ones who turned up their noses at small-fry developers, racing to get their own app stores into production.

As the biggest break from convention, the App Store isn't without controversy, mostly involving the opaque review process that all apps must undergo before appearing in the store. A slow approval process has frustrated developers trying to get bug fixes to users, who post negative reviews and complaints about problems that already have fixes done and awaiting release. In an event earlier this year, Apple crowed that 98% of apps were approved in 7 days or less. But the more significant complaints about store reviews have involved the fairness of the process, as well as the competence of reviewers. In one recent controversy, developers have complained that Apple is now requiring them to apply an "adults only" rating to apps that can connect to the internet, and therefore access potentially illicit content. By this logic, the iPhone's bundled web browser, mail app, and iTunes player should all be similarly rated and restricted. Other developers have had apps rejected for unwittingly duplicating functionality of upcoming software updates, or providing functionality that is merely similar to Apple-provided apps.

The other challenge with the App Store is one of marketing: with tens of thousands of apps in the store and more arriving every day, it's hard to stand out. If the App Store interface is the only way for users to find an app, it leads to a battle to get visible in that interface, something that many have tried to do by gaming their prices to get into the "Top 25" tab, creating a downward price spiral that hurts all developers. But the App Store isn't the only way to get noticed: it's easy to provide a web link that takes browser users to the App Store (either on their iPhone, or via iTunes on Mac and Windows). As the iPhone platform continues to grow, it's likely that more and more people will discover apps through means other than the App Store itself. At least that's my fervent hope, because there are whole classes of applications that won't get written if developers can't charge more than a dollar or two for them.

Be Different

In many ways, the iPhone goes against a lot of the recent trends in software development, conventional wisdom we'd taken for granted. While we've spent much of this decade learning scripting languages, praising Open Source, and predicting that everything would eventually turn into webapps, the iPhone SDK revolution arrived in the form of C-based native applications on a highly proprietary and closed platform. That it's successful at all should give many of us pause to rethink our assumptions and beliefs: were webapps only popular because they didn't have to be installed? Did the App Store solve that?

Now that the iPhone is the most talked about platform of the day, it might make you wonder if you should jump on board. The price of admission is low. To run the SDK, you'll have to use an Intel-based Mac, reflecting the high level of code sharing between the Mac and iPhone. The SDK and basic membership in Apple's iPhone Developer Program is free, allowing you to get your feet wet developing apps and run them in the included iPhone Simulator. When you're ready to test your code on a device, you need to purchase either an iPhone or an iPod Touch, and upgrade to a paying developer membership (currently $99 for individuals, $299 for corporations). With your paid membership, you'll be able to generate app-signing certificates to put code on your device, and to submit apps to the App Store.

We've heard a lot in the media, and from Apple's own PR, about "get rich quick" stories, developers who plugged away on iPhone code at night, hit the jackpot, and quit their day jobs to tour the world. Entertaining as these anecdotes are, I'd like to encourage you to set them aside and consider the merits of iPhone programming as an end in itself. If you're, for example, a server-side middleware developer who works in Ruby or Perl, the experience of writing user-facing mobile applications in Objective-C couldn't be more different than what you're doing now. As an intellectual exercise alone, an exposure to Cocoa's patterns, Interface Builder's elegance, the challenge of working with significantly limited resources, and the sheer fun of getting a whole lot of cool into a tiny little app is probably irresistible to many developers.

Perhaps more importantly, the emergence of the platform and the popularity of quirky third-party apps means that the indie developer is back. And with luck, he or she can actually pay the bills by working on a platform that's fun and intellectually rewarding.

We've had a year to get the 99-cent "ringtone apps" out of our system, and every week we see remarkable new ideas take flight in the App Store. And this is only the beginning. Declaring that "there's an app for that" is premature; there are lots of great ideas that don't have apps yet. Will you write one of them?

About the Author

Chris Adamson is a writer, editor, developer and consultant specializing in media software development. He is the co-author, with Bill Dudney, of the forthcoming Pragmatic Bookshelf book iPhone SDK Development [U1]. He has served as Editor for the developer websites ONJava and java.net. He maintains a corporate identity as Subsequently & Furthermore, Inc. [U2] and writes the [Time code]; [U3] blog.

External resources referenced in this article:

[U1] http://www.pragprog.com/titles/amiphd/iphone-sdk-development

[U2] http://www.subfurther.com/

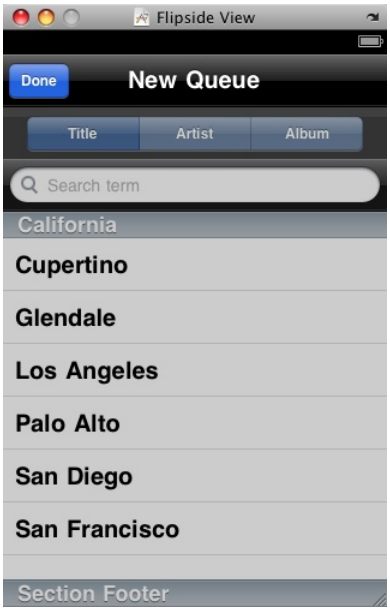
[U3] http://www.subfurther.com/blog

iPhone, Meet Cucumber

Testing Your iPhone Apps with Cucumber

by Ian Dees

It's the hot development platform, and anything that speeds the process of getting your iPhone app finished and tested is welcome news. So Ian Dees shows how to drive an iPhone GUI from a Ruby test script.



It's been a busy year since the first official iPhone SDK arrived. In the land rush to the new platform, developers have found creative ways to test their apps. The SDK ships with debuggers and profilers, of course. But there's also been a groundswell of home-brewed recipes, from cross-language unit tests to the beginnings of GUI automation.

In this article, we'll sketch out one path to driving an iPhone GUI from a Ruby test script. If you'd like to try the examples out on your own Leopard machine, the full source code is available[U1]. You'll need the iPhone SDK and simulator, of course, plus a few Ruby libraries:

```
$ sudo gem install cucumber rspec tagz
```

When you have the luxury of a full desktop OS, you can often simulate user interactions like button presses with just a few API calls. With the iPhone, there are some extra steps. It's worth taking a peek at these low-level tasks, before seeing how to assemble them into a whole test.

Simulating Events

The centerpiece of the iPhone user interface is the touch screen, so we'll start our exploration there. The iPhone OS passes your app a mix of UIEvent and UITouch objects to represent a tap on the screen, but doesn't offer direct access to those objects' details. To simulate a touch event, you need to set up a bunch of private, undocumented fields in just the right way.

Matt Gallagher, a software developer and prolific Cocoa blogger, has blazed this trail for us. He's added handy setup methods to the relevant Objective-C classes using a category.[7] This took a lot of careful observation and experimentation on his part, so all your automation project needs to do is call one method on his ScriptRunner object:[U2]

```
[self performTouchInView:someUIObject];
```

The other half of the equation is interrogating the user interface to find out what happened after that screen tap. Again, Objective-C categories come in handy here. Matt's fullDescription method, added to every UIView object, returns an XML string representing a user interface element and all its children. (See this article[U3]) Here's what the description of a simple button might look like:

7. Objective-C's civilized take on a monkey patch—a modification to an existing class.

```
<UIButton>
  <address>17252816</address>
  <tag>0</tag>
  <currentTitle>Home</currentTitle>
  <frame>
    <x>0.000000</x>
    <y>0.000000</y>
    <width>65.000000</width>
    <height>31.000000</height>
  </frame>
  <subviews>
    <!-- elements inside this button... -->
  </subviews>
</UIButton>
```

Matt's original SelfTesting iPhone app included these two methods—performTouchInView: and fullDescription—and a few convenience methods. Felipe Barreto's Bromine project [U4] project takes these, adds a few more, and wraps it all up in a package designed to be dropped easily into an existing iPhone project.

Driving the App from a Script

How does Bromine know what to do and when? At app launch time, the ScriptRunner object runs a series of instructions from a property list file (parsing this format is a simple library call). These instructions use XPath to describe a GUI object's location in the hierarchy of user interface elements.

Here's a simple TestScript.plist that searches the current screen for a UIButton with the title Home, and clicks on it.

```
<plist version="1.0">
  <dict>
    <key>command</key>
    <string>simulateTouch</string>
    <key>viewXPath</key>
    <string>//UIButton[currentTitle="Home"]</string>
  </dict>
</plist>
```

It's easy enough to get going with a simple plist file and a few test steps like this one. But let's consider a twist. We can teach Bromine to listen for instructions while the program is running, rather than reading a static file once.

The sample code for this article includes a modified version of Bromine that accepts test steps handed to it by an embedded web server.[11] This approach frees us from having to stop and restart the app while we're writing tests. It also lets us drive the app from the command line, using tools like curl:

```
$ curl -d @TestScript.plist http://localhost:50000/
```

Now the doors are open to connect your app to any test harness you like.

11. This project uses cocoahttpserver [U5]..

Writing Tests in Cucumber

Speaking of test harnesses, it's worth taking a second to introduce Cucumber [U6], the framework that encourages writing tests in plainspoken language. Imagine we're building a blogging app. Here's how we might express a simple task in Cucumber:

```
Feature: blog posting
  As a blogger
  I want to post from my iPhone
  So that I don't need to drag my laptop everywhere
  Scenario: short posts
    Given the blog "example" with user "me" and password "secret"
    When I add a post entitled "First post!"
    And I add a post entitled "Shark jump"
    Then the blog should have the following posts:
     | title       |
     | Shark jump  |
     | First post! |
```

It's plain English, but it's also runnable code. So you can use scenarios like these to describe how an application will behave, and then later use them as the automated portion of your acceptance tests. Of course, a real project would need lots more test cases. But this one will keep us plenty busy for now.

Trying it Out for Real

Cucumber can serve as a design tool, helping coders and customers agree on features before their product even exists. But for the sake of trying the techniques in this article, it turns out there's already a real blogging app we can bounce our script off of: the open-source WordPress iPhone client [U7]. It's easy enough to take the stock source code, drop in Bromine to drive the GUI, and add a web server to listen for instructions. You'll find all the pieces assembled for you in the source code for this article.

So how do we drive the WordPress app from the top-level script? Cucumber uses regular expressions to match each step of the plain-language test to a chunk of Ruby code implementing that step. Here's what the first of those steps looks like; it clears out the list of blogs, and then adds a new one. Keep in mind, the `Blog` class doesn't exist yet; we'll get to that in a second.

```
Given /^the blog "(.*)" with user "(.*)" and password "(.*)"$/ do
  | blog, user, password |
  Blog.empty!
  Blog.add \
    :host => "#{blog}.wordpress.com",
    :user => user,
    :pass => password
end
```

The `Blog` object is going to construct pieces of XML and use HTTP POST requests to send them to the iPhone. We'll use the Tagz library to write Ruby code that mirrors the structure of the XML. Here's how we'd push the Home button from Ruby.

```
xml = Tagz.tagz do
  plist_(:version => 1.0) do
    dict_ do
      key_    'command'
      string_ 'simulateTouch'
      key_    'viewXPath'
      string_ '//UIButton[currentTitle="Home"]'
    end
  end
end
# Like Ruby's built-in post_form, but with text
# instead of form fields.
Net::HTTP.post_quick 'http://localhost:50000/', xml
```

Our Blog class members won't be building XML and sending HTTP directly. Instead, they'll lean on wrappers that do the grunt work of pushing a button or filling in a text field:

```
BlogSettings  = '/./descendant::UIButton[1]'
RemoveBlog    = '/./descendant::UIRoundedRectButton'
ConfirmRemove = '//UIThreePartButton[title="Remove"]'
def Blog.empty!
  count.times do
    press BlogSettings,
          RemoveBlog,
          ConfirmRemove
  end
end
```

The rest of Blog's methods are similar combinations of screen taps and text entry. To keep the pace quick, let's gloss over those and move on to the final test step:

```
Then the blog should have the following posts:
 | title       |
 | Shark jump  |
 | First post! |
```

How does Cucumber handle that ASCII-art table of blog posts we're expecting to see? Here's the accompanying step definition in Ruby:

```
Then /^the blog should have the following posts:$/ do
  | posts_table |
  Blog.first.posts.should == posts_table.hashes
end
```

For this style of step definition, Cucumber passes the entire table as an array, one item per row, inside the posts_table variable. As with a spreadsheet program, the first row is a guide to the remaining rows. Here's what the resulting Ruby array looks like for this example:

```
[{'title' => 'Shark jump'},
 {'title' => 'First post!'}]
```

So the Blog#posts method just needs to make sure it returns all the blog posts in the same format, so that the should == expectation can compare them and report a passed or failed result.

Looking ahead

Where do we go from here? There are tons of possibilities.

You've probably noticed that the examples here have had a lot of typical real-world robustness measures left out for the sake of brevity. Adding a few checks to make sure we're on the right screen when the test begins, and switching from fixed delays to smarter waits, are logical next steps.

Another promising direction would be to increase the vocabulary of events we can simulate. There's a world of input methods out there for the taking, from some of the more automation-resistant text controls to things like the accelerometer.

Finally, it would be nice to expand this demo beyond the simulator and into real hardware. Imagine your build server doing an automated smoke test on a live iPhone after every source code check-in.

I hope that what you've seen so far has whetted your appetite to explore further. Please download the source code for this article, try the examples, and maybe even add a test scenario or two. Drop in on the discussion forums [U8] if you have any questions or comments. Happy coding!

About the Author

By day, Ian Dees slings code, tests, and puns at a Portland-area test equipment manufacturer. By night, he dons a cape and keeps watch as Sidekick Man, protecting the city from closet monsters. Ian is the author of *Scripted GUI Testing With Ruby* [U9], published by the Pragmatic Programmers.

External resources referenced in this article:

[U1] http://www.bitbucket.org/undees/cucumber-article

[U2] http://cocoawithlove.com/2008/10/synthesizing-touch-event-on-iphone.html

[U3] http://cocoawithlove.com/2008/11/automated-user-interface-testing-on.html

[U4] http://code.google.com/p/bromine

[U5] http://code.google.com/p/cocoahttpserver

[U6] http://cukes.info

[U7] http://iphone.wordpress.org

[U8] http://fora.pragrog.com

[U9] http://pragprog.com/titles/idgtr

Pragmatic Life

Andy Hunt Talks about Life and Everything Else

interviewed by Michael Swaine

Dave Thomas and Andy Hunt *are* the Pragmatic Programmers. Last month we talked with Dave about publishing, and this month we chat with Andy about the Pragmatic Life book series and about life in general.

Writing code is just a means to an end. It doesn't define us.

ms: Andy, when we were planning *PragPub*, you told me you wanted it to be a place where we can share our passion for what's cool. The idea, I think you said, was to provide a peek at what we're excited about and what our authors and readers are excited about—and we shouldn't be shy about proselytizing, we should try to get our readers excited about things we think they should be excited about, so we can rediscover important truths about our profession and our craft.

ah: Sharing the excitement is an important part of it. We get a fair number of proposals from people every week, so we really get to see what folks are excited about. Some of the topics are well-suited for books that we might publish. But not all of them are; sometimes it's just a really cool topic that people happen to be interested in.

You can tell when the author is passionate about their subject. If they're not, you tend to get a very dry and dull proposal, which would make a dry and dull book. They're just trying to explain the topic, so the work ends up reading a bit like an encyclopedia entry. Not very exciting, illuminating, or engaging.

However, for topics that people are genuinely passionate about, there's a very different feel. You can sense and share in the excitement. It's the difference between being shown an exploded parts diagram of an antique motorcycle, versus a friend roaring up in a cloud of dust, engines pumping, who shouts out to you, "Hey, hop on the back of my ride." That's the feeling we're aiming for.

ms: And I appreciated it that you said our scope shouldn't be limited to programming. Because there's more to life, and to a programmer's life, than code.

ah: Oh absolutely. Writing code is really just a means to an end. It's a tool. It's a tool we happen to use a lot of, and tend to prefer, but it doesn't define us. Too many people I think identify themselves as a Java programmer, or Ruby programmer, or PHP programmer, or what have you. But in reality, that's not what we actually do for a living. What we do is solve problems.

ms: With that in mind, let's talk about life.

ah: In the immortal words of Jethro Tull, life's a long song. Things change. Your career focus when you're just starting out is going to be very different than in the middle of your career, and quite different yet again towards the end of your career. Everything changes over time.

You're a professional, and you're good at what you do.

ms: Why, thank you.

ah: I'm talking to the reader...

ms: Got it.

ah: You're good at what you do, you've probably been doing this for a while, and you're on your way to the top of your profession. You've spent long, occasionally happy hours, solving intricate problems using code. You've resolved thorny issues, tracked down elusive bugs, and finally figured out what the chaotic users really wanted in the first place. There's a great sense of accomplishment.

But somewhere along the line, you come to realize there's more to life than code, more to life than your day job. You start to change your focus, to take a wider view.

There's digital photography, robots, music, health, career, family—all the other topics that we're interested in. Our hobbies and passions might include coding. Maybe your projects at work aren't using that shiny new language you'd like to try, or maybe you want to play with a more sophisticated operating system than is in general commercial use. Or maybe you want to actually get away from the code altogether, and explore a hobby where technology is tangential to the task at hand.

Even though these extracurricular activities aren't part of your day job, you can still apply the full force of your pragmatic powers to learn and enjoy them. You can use the same skills you use on the job—quick learning, applying feedback, making pragmatic choices—in your hobbies or other personal areas.

ms: I'd like to crank up the focus really high. I want you to talk about *your* life. Your hobbies, your passions. What gets Andy Hunt's blood pumping?

ah: *World building*. I think most creative endeavors take on some form of world building. One of the neatest parts, to me, about writing code is creating that little world, that little environment in which it lives. You are defining your own laws of physics, your own social contracts, your own relationships between entities you have devised, and it's a lot of fun. But of course there are other activities where you can build worlds as well: Fiction writing, role-playing games, composing a rock opera, restoring an antique car, or creating a photograph with unusual lighting from an unusual angle to show part of the world that no one else has seen—or that doesn't even really exist.

I enjoy woodworking as a hobby. It's a nice change of pace to do something very tangible and physical after twiddling bits all day. But my real passion lies in music. I've performed live, I've written and recorded songs, I've practiced my instruments, and enviously eyed new ones. I'm always humming or whistling something. It's just part of my mental fabric. I enjoy creating songs that envelop the listener, that draw you in and surround you. Perhaps they are complicated, perhaps they're very simple, but I always try to make them immersive.

ms: For me that raises a question I've pondered more than once: why are so many programmers also musicians?

ah: I think there's a high degree of correlation there. There are some studies that show that music is one of the few activities that actively engages almost all faculties of the brain simultaneously, so it's a very stimulating experience.

And certainly if you're writing your own music, or are immersed in jazz improvisation, or performing a challenging technical piece, you are building your own little world. Same as coding. So it's not at all surprising to me to find a great deal of affinity for music among programmers.

ms: I'm sure it doesn't surprise anyone reading this that you have these rich extracurricular interests. These passions. And that's really the only word, isn't it, that captures the range of interests and activities that absorb us when we're not coding. It's not just hobbies. There's family. There's your career, apart from the actual coding. We all have these absorbing passions, and they are part of who we are.

ah: Sure—it's what makes us human. I love talking to folks at conferences, because there you really get a chance to find out the *other* stuff that people are into. Maybe it's related to music or photography, or perhaps they're building a rocket sled in the backyard. Or a potato cannon. My programmer friend Lou Bottali built a floating one-man submarine out of 55-gallon drums.

If you get people talking about the stuff they're passionate about, you can really see their eyes light up. It's obvious they're excited about it. Sometimes much more so than the latest 1.6.7.9 release of whatever.

ms: And yet there is this stereotype of the computer geek with no life. Oh, I know that being smart and being technically astute are now cool. But the stereotype hasn't really disappeared. Programmers are still being advised to get a life. In fact, isn't it true that you considered calling the Life line of books, Get a Life?

ah: Yes, and I certainly still think of it that way. Even though the formal title is *Pragmatic Life*, the intent is to get a life. I think the stereotype still exists for some, and remains a cautionary tale for all. Programming is a demanding discipline. It's very easy to become involved with the

project that commands all of your available time—and then some. It can literally suck the life right out of you.

And that's not just a metaphor. Stress and overwork have been shown to rob you of creative cognitive abilities, and there's a recovery lag before you get them back. Your brain is wired such that you need to daydream. You need to do something kinesthetic, using your hands. You need to shift your focus and concentrate on sophisticated scale patterns, or complex needlepoint designs, or whatever.

ms: So does the book line span all of these non-coding passions? I know that it includes career-enhancing books, like Andy Lester's Land the Tech Job You Love [U1] and Chad Fowler's The Passionate Programmer [U2], and also books on topics that might be considered hobbies.

ah: That's our plan. We are looking at topics ranging from photography, to music, to yoga, to electronics. We started off with the career-oriented books, but will include hobbies—both mainstream and the more unusual—and titles geared toward family and personal development as well.

ms: It might be thought that the career books are more important or central to your mission, but I think you have some thoughts about the importance of hobbies. Certainly one could argue that these other interests lead to a more balanced life, but you think they are more important than that, if I understand correctly.

ah: Much more important. We tend to think of work life as separate from our hobbies, as separate from our families or social network (i.e., "friends"). But the distinction is artificial.

The discipline of *Systems Thinking* encourages us to view the world as a series of interacting, interrelated systems. While this is common enough when talking about project teams and organizations, it's easy to forget that each one of us is a system, too. Not only do we function as part of larger social and technological systems, but taken individually each of us is a collection of systems as well. And because everything is interrelated, then our hobbies, passions, and families have a direct effect on our professional life—and vice-versa.

Sometimes the link is obvious. I tend to talk about learning quite a bit; learning is obviously critical to our career success. But it's important to keep learning throughout your life. When you're given a new and novel task to learn, your brain experiences increased activity and neuron growth. By learning more, you'll literally develop more brainpower to learn even more—whether on the job or off.

The controversial historian Arnold Toynbee claimed that civilizations waxed and waned through distinct, predictable stages. On the path to decline and fall, they'd pass from abundance to selfishness, then down through complacency to apathy, sliding inexorably into dependency, and finally into bondage.

I don't know if that's true for civilizations (that answer would be "above my pay grade," as they say). But it does seem to have a ring of truth about people and our careers. Selfishness disrupts your social network; complacency leaves you vulnerable to fast-moving changes in technology and markets; and apathy becomes the mental paralysis that renders you incapable of response. You're left so dependent on the existing status quo that you become a slave to it. That's not a pretty sight. But it is avoidable.

On Toynbee's "up" side, faith and courage leads to liberty, and then to abundance. And you can strive towards abundance by expanding your reach—beyond code, beyond the day job. Have the courage and faith to extend yourself. Your extracurricular activities can have a very positive effect on your career in a number of non-obvious ways, from increasing your overall skill set to expanding your social circle, from providing rich metaphors for system behavior to supplying fertile ground for a sudden mental breakthrough.

---

ms: Anyone who has read The Pragmatic Programmer [U3] knows that you advocate a pragmatic approach to coding, and that you have a pretty well articulated view of pragmatism. You've been teaching me to think in pragmatic terms about what I do, and I gather you advocate a pragmatic approach to these extracurricular passions as well.

ah: Naturally :-). But pragmatic doesn't mean dull or tedious—we do this stuff because it's fun. Dave is into photography; he's got a wicked zoom lens that looks like a small bazooka. One of these days I'll make good on my threat to release an album. Our editors and authors enjoy a surprising variety of hobbies ranging from yoga and Taekwondo to needlepoint and craft brewing.

They really do this stuff, and you can too. There are plenty of other books and magazines out there that cater to these topics, but they're just not, well, *pragmatic* about it. I enjoy woodworking, but many of the high-end woodworking magazines just seem to emphasize the point that I will never successfully spend 300 hours hand-carving fretwork to make a coffee table that looks like a lace doily. Even many of the geek-culture projects fall prey to a similar lack of pragmatism. Not all of us have welding supplies, eprom burners, or laser cutters close at hand. The idea behind the Pragmatic Life series and the PragPub magazine is, after all, a pragmatic and practical focus.

Our approach is that you *can* do this, and it *will* help your career and your overall life.

---

ms: If someone wants to pitch a book idea for the series, what should they do?

ah: Contact us! Check out this link [U4] for details. It explains what you need to submit, but more importantly it explains what you get and what the experience is like writing with us. As everyone is fond of pointing out, we are not like most publishers.

In fact, we are not really "publishers" at all. We're a couple of programmers who happen to publish books, on topics we'd like to see.

ms: If the other books are for professional skills, are these about amateur skills?

ah: I suppose it depends on what you mean by amateur. I'm an amateur at my hobbies—in the original sense of the word. Unfortunately, the shifting sands of the English language have turned the word around to mean inept, or unskillful. But the root of amateur comes from the Latin for love, meaning that we do this not for money or fame, but because we love it. Sometimes we can get paid for it as well, but that's just gravy.

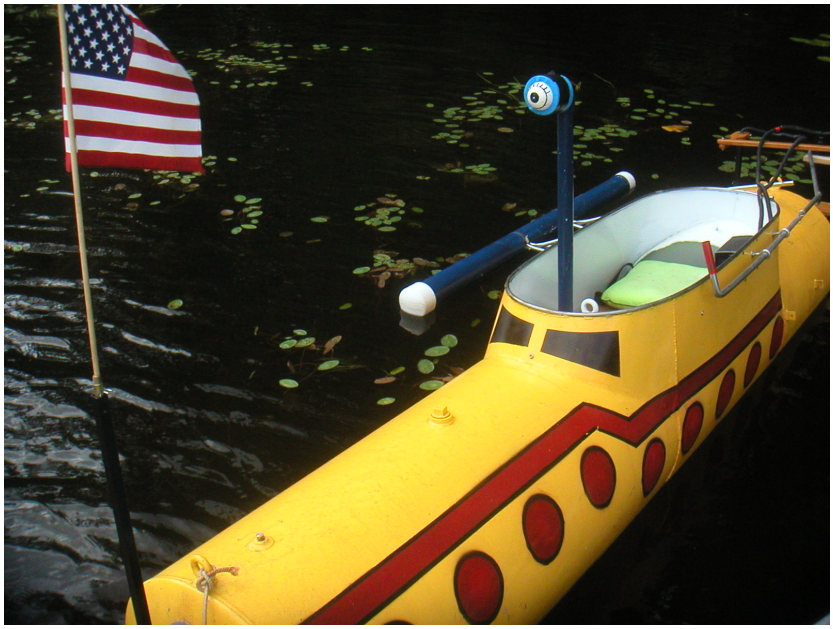
We're all amateurs at technology, in that sense. We embrace it because we love it. We love to learn, to fiddle, to create, to build worlds. We do that all the time in our professional lives, but I feel strongly that we need to extend that, and do so in the rest of our lives as well.

Because life doesn't stop at the office door. And neither do we.

About the Author

Andy Hunt is one of the Pragmatic Programmers.

(P.S. And since I know you're curious, here's a picture of Lou's sub:)



External resources referenced in this article:

[U1] http://www.pragprog.com/titles/algh/land-the-tech-job-you-love

[U2] http://www.pragprog.com/titles/cfcar2/the-passionate-programmer

[U3] http://www.pragprog.com/titles/tpp/the-pragmatic-programmer

[U4] http://www.pragprog.com/write-for-us

Managing Your Life Projects

Surprise! You're Making Dinner Tonight

by Johanna Rothman

The same techniques that make you effective on the job can save you time at home and enrich your daily life.

I'll bet that at some time on a job you've been told you need to "do more with less." Or to "work smarter, not harder." Or—one that irritates me no end—"be more productive."

Well, I don't try to be *less* productive! In fact, over the years I've learned a few things about how I work, and how to be as effective as I can be. And while it's no fun being *told* to be more productive, there is a lot of satisfaction in learning techniques to help *yourself* do more with less, work smarter, and be more productive.

But you don't have to leave your productivity insights behind when you get up from the keyboard. As you'll see, the tips I'll share here are just as useful in tasks I perform outside of my profession, saving me time and even making the tasks more fun. I hope you'll find that they make *your* life a little more productive and more enjoyable, too.

Know What You Have to Do

Here's a simple rule that, consistently applied, saves me all sorts of grief. I write everything down that I have to do. All of it. On a list. Just that. I may use cards or stickies to get started, but I make sure everything is one place. As you collect it, merge everything into one list. That's *one* list. Having everything in one place is necessary—not several todo lists, several backlogs, several stickies. Just as an agile project can have only one product backlog, you need one todo list.

Yes, I know what you're thinking. If you're one of those people whose todo list is longer than both arms and legs, don't worry. Just start writing things down, all in one place. When you're done writing things down, for the moment, stop. If you remember something later—dentist appointments are always the thing I forget to write down—add it then, when you remember it.

You Can't Multitask

You can't multitask. You think you can, you think you *do*, but really, you can't. Handle one project at a time. Forget the idea of multitasking on several projects at once. You can't do it. At least, you can't work on several projects and still be effective. You can, however, have several tasks in service for one project. The smaller those tasks are, the easier it is to finish one and decide which piece of work to pick up next. The key is finishing a piece of work before you move to another piece of work. You can't multitask.

This is why you need that one list. Once you know what you have to do, you can select one project to work on.

For example, if you make dinner, you have one project: get all the food on the table at the appropriate temperature at the same time. If you make chicken, rice, a salad, and green beans, you will have to start and stop the different tasks to make sure that what you end up with is dinner and not dining disaster. However, if you have a large enough kitchen and several people on your dinner team, you can assign each a piece of the project, and start each individual task depending on how long it will take.

Although I claim you're not multitasking, each member of the team will probably have to handle several small tasks, focusing on one task until that task is done *enough*. For example, the chicken has to be cooked for longer than the the other parts of the meal. That means Daughter #1 can start and finish the "prep the chicken and stick it in the oven" task, even though that doesn't mean she's finished cooking the chicken. She may have some basting to do later, but she can do that between making the salad, setting the table, and maybe even buying flowers for the centerpiece. The key is to start and finish a small piece of work before you move to another piece.

Work Until You Get to a Wait State

Once you have these small chunks of work, you're able to work on one of them until you get to a wait state. Sometimes, that means you have to rethink how small a chunk you're working on.

In the dinner example, if you want to have a salad of lettuce, red cabbage, black olives, artichoke hearts, celery stalks, sliced tomatoes, carrots, and cucumbers, Daughter #2 has a lot of cutting ahead of her. Instead, if you have very small tasks such as:

- Get all veggies for salad out of the refrigerator
- Get all cans for salad out of the pantry
- Cut cucumbers, integrate
- Cut lettuce, integrate
- Cut red cabbage, integrate
- Cut celery, integrate
- ...

I started doing this when my children were very young, because we had predictable nightly crises while I was trying to cook dinner. With small tasks, it was easy to know where I was. Later, when the children were older, I could put these practices into play as I put the children to work. And whether or not the predictable crisis materializes, I have much more predictability in how long I will need to finish making the salad.

Back to your dinner prep: If you do have other people available to help make the salad, you can ask for help. I'll ask, "Patty, can you cut the cucumbers? Sue, can you cut the red cabbage? Don, can you chop the celery?" As each person finishes his or her tasks, it's easy to see what tasks are left. If Patty, Sue, or Don has time to do more, she or he can.

Working until you get to a wait state requires very small atomic tasks. The task isn't "make dinner" or even "make salad." Instead, the task is cutting a particular vegetable and integrating it into the salad. Yes, continuous integration works for salad just as well as for code.

Refactor What to Work On

The nice thing about setting small tasks and completing them is that you get to choose what to work on next. You can scan your list (a la Mark Forster's Autofocus system [UI]), and see which tasks to start—and complete—next.

If you have the chicken cooking away and the salad cut up, you have some choices about the rice, the vegetable, the salad dressing, maybe even dessert. Maybe you'll take the time to baste the chicken now. But you get to choose. You've bought that for yourself.

Other Techniques I Use

You may not have much experience with collecting your work or creating small, atomic tasks. That's OK. Just use a timebox to practice.

What's a timebox? It's this: You say "I'll work on *this* task until *that* specific end time" and then you stop at that time. Timeboxes provide you with a focus and a way to look back at what you've been doing to see if you've been successful.

Timeboxes can help you succeed with something new where you're not sure if you really know what you're doing. They're also great if you dislike the task you want to complete. Sometimes, if you just commit to it for a timebox, the task is easier to complete, and then it's off your list. For example, I hate cleaning my office. But every so often, it's messy enough that I need to reorganize. Because I hate spending time on this task, I timebox it to an hour at a time. Once my time is up, I get to choose again if I want to spend more time on it, or if I've had enough. You are always, when it comes to getting the job done, your own boss, because only you can make you do it. You can decide whether to hire yourself by the hour or by the task.

Did That Help?

Here's a recap of the tips:

- Collect all your work. Capture all the outstanding work in one place, written down. I prefer list form.
- Work until you get to a wait state. Create small chunks of work, and complete one of those before you move to another.
- Create small, atomic, independent tasks. Finish one before you move to another.
- Use timeboxes to practice new approaches. Learning something new is not easy. It's easier when you try it for a little while and look back and see how you've done.

Do you need to be more productive in life? I'll bet you do. I know I do. I like knowing that I can be productive and efficient in tasks like cooking dinner, so as to free me up to do other more interesting things, such as reading or

ballroom dancing. Or writing. I hope these simple techniques help you as much as they help me.

About the Author

Jolt Productivity Award-winning author Johanna Rothman helps leaders solve problems and seize opportunities. She consults, speaks, and writes on managing high-technology product development. She enables managers, teams, and organizations to become more effective by applying her pragmatic approaches to the issues of project management, risk management, and people management. Johanna publishes *The Pragmatic Manager*, a monthly email newsletter and podcast, and writes two blogs: "Managing Product Development" and "Hiring Technical People." She is the author of several Pragmatic Bookshelf books, including the forthcoming Manage Your Project Portfolio: Increase Your Capacity and Finish More Projects [U2].

External resources referenced in this article:

[U1] http://www.markforster.net/autofocus-system/

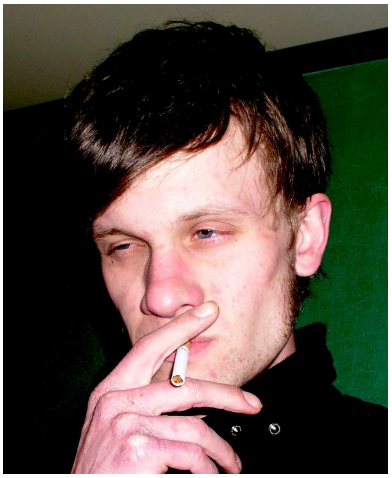
[U2] http://www.pragprog.com/titles/jrport/manage-your-project-portfolio

Shady Illuminations

Anti-social Software

by John Shade

John confronts his inner nerd, who turns out to be just as misanthropic as his outer nerd.



It's contrarian to argue with success, which is an excellent reason for doing it.

Apple's I'm a Mac/I'm a PC commercials are highly watchable, but I say they fail at what should be their sole purpose: promoting Apple. The character we're supposed to identify with is less engaging than the character we're supposed to laugh at ("PC"). In the commercials, "PC" is Lou Costello, "Apple" is Bud Abbott. Apple is Rowan to PC's Martin, or Martin to PC's Lewis. PC is Tommy, Apple is Dick; PC is Cheech, Apple is Chong. You get the point: PC is more interesting, gets all the good lines, and stays in our minds after the commercial is over. The ads are successful, I guess, but what they're successful at is getting us to identify with a PC.

At least those of us who identify with John Hodgman.

John Hodgman is so successful in the role of PC partly because he's a gifted comedian, and partly because the role isn't that much of a stretch. Like John Wayne, whom he resembles in no other respect, Hodgman mainly plays himself. He's a nerd.

It's because he's a gifted comedian (and writer) and because he really is a nerd that he was able to pull off that highly complex performance at the Radio & TV Correspondents' Dinner roast of President Obama. You know, proving that Barack Obama is our first nerd President, then undoing his own argument with hints that Obama might actually be a jock. ("Despite his Spockish calm and gangly frame, the President is known to dabble in sports... and not just bowling, but the hard stuff. What they call 'basketsball'.")

It was a tour de force performance. It demonstrated Hodgman's comedic genius and pretty much spiked any suspicion that his nerdiness is just a role. Hodgman is a nerd, a math geek, a wimp. You just know that bullies kicked sand in his face at the beach when he was a teenager. Heck, listening to his weedy, nasal voice, *I* want to kick sand in his face, and I don't even like the beach.

But I found myself getting more and more uncomfortable watching the performance, because it brought to the surface some of the creepy misgivings I have about the direction of software development today. Because in an uncertain world, I hew to the eternal verities: software developers are social misfits and users are lusers.

Excel Is a Database. Deal with It.

Apparently all software is social now. As usual, I must have been hanging out in the parking lot when the announcement was made. I have to say, it would have bemused me to hear that announcement. I am about half-bemused now, because nerds don't do social.

Nerds who develop software (but I repeat myself) do deal with people, of course. They even deal with users, if only remotely. Because you have to, don't you? If you make any attempt at all to make software that is really used, you have to spend time in the intellectual sewers of actual users' minds.

Which is where you discover that Excel is a database. It's an established fact in Userville, so just accept it. As Alan Cooper said, the inmates are running the asylum. He was talking about you, not users, but he got the lyrics right even if he screwed up the tune.

You can accept that Excel is a database to users, but you don't forget that it's *really* a spreadsheet program. It's just that—put it this way: I applaud the fact that Sonia Sotomayor has empathy. In every interaction I've ever had with a judge, empathy is just what I thought they didn't have nearly enough of. And when the judge seems to display a disturbing familiarity with nunchucks and their potential for the infliction of injury, I say the more empathy the better. But empathy for users is an alien concept in the nerd psychology.

Adam Boswick says that it's the software that is simple, sloppy, and forgiving of human foibles and weaknesses that lasts. Well, that just sucks. If you didn't have to put up with the stupid user, you would never in a million years write software like that. You'd write tricky, tight, rigidly-designed hacks. At least you would if you're a proper nerd.

It used to be enough to make the software work. But when software is all about human-human interaction, the goal becomes to make the human-human interaction work. And it's worse than that, because social software is not about individual users. You have to understand groups, which, it turns out, can't be done by understanding an individual user and iterating.

Clay Shirky defines social software as "software that supports group interaction." It isn't social software until you have three people interacting. Email is Cribbage, social software is Bridge. (Which would make porn Solitaire.) And it's at that group level where things get interesting. Groups have their own dynamics. Emergent properties, er, emerge.

You might think that this is nothing new. You might say that Second Life is just a remake of Lambda Moo with modern special effects. I wouldn't argue with you. But what is different today is the ubiquity of social software. Facebook has more members than many countries have citizens. We're no longer in a time of ten thousand-member groups, when social software cheerleaders spoke of a "global village." Although it's still a village if you define "village" as a place that enables gossip and vendetta.

Historically, social software has been designed to facilitate flame wars. Seriously, studying these programs objectively, you would have to conclude that this was at the top of the requirements list. Because they do it so well. Even a mild-mannered Wiki is as much a battlefield as a Listserv. It's just a paintball field where the guns are loaded with turpentine.

All I can say is, it's a good thing that these crowds will all be wise and these mobs will all be smart. It was a smart mob, we learn, that brought down Philippine dictator Ferdinand Marcos. That movement provides this example of how a smart mob communicates:

Go 2EDSA, Wear blk.

Deposing a dictator may be commendable, it is surely difficult and it's probably heroic, but while I mean no disrespect to the heroes who gathered in the Philippines, no amount of bravery can turn a flashmob into a brain surgeon. Smart mobs may happen, but when they do they're news. You can't sell a book on the stupidity of crowds for the same reason you can't sell a news story titled "Dog Bites Man." Stupid isn't news; it's the default setting. More often than not, the village *is* the idiot.

In July the first nerd President used the word "stupidly" in characterizing the Cambridge police's handcuffing of a Harvard professor for getting snippy with an officer. A couple of days later Jeff Bezos used the word "stupid" to characterize Amazon's unpublishing of George Orwell on people's Kindles. It's gratifying to see "stupid" getting better shelfspace in the marketplace of ideas.

R U a Nerd? U R not Social.

If you develop software, you're supposed to be a nerd. And there are clear rules dictating how you should behave. For example:

Joel Spolsky: "Silicon Valley nerds getting venture capital to codify their own Asperger's Syndrome... and demonstrate thoroughly just how completely they don't understand human-human interaction." Research supports Spolsky, suggesting that autism, Asperger's Syndrome, and Attention Deficit Disorder are rampant throughout Silicon Valley. Yeah, baby, I want to be an ADD multitasking ninja, because all the cool kids are doing it.

Po Bronson reports that the co-founder of Yahoo used to sleep under his desk when he was worth half a billion dollars. A role model for teen-aged nerds everywhere.

Paul Graham wrote a book. It seemingly went over some readers' heads. "I didn't want to waste people's time telling them things they already knew," he explained. "It's more efficient just to give them the diffs."

Having sand kicked in face by computer priesthood led to rebellion and successful rebels often become bullies. Alan Cooper says nerds have a tendency to act like jocks, snapping emotional towels and administering mental wedgies to users.

Richard Stallman, the most dedicated defender in software development creates lists of "words to avoid." Disdain for the priesthood seldom survives ordination.

Danah Boyd points out the size of the gap. Only in a nerd's mind is it an acceptable solution to the breakdown of comity in online social space to suggest that the user just adopt multiple personalities. Hey, my psychological disorder's working well for me, maybe you should get one too.

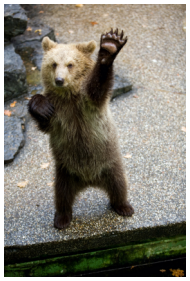
Cooper offers this test for nerdiness:

- Have you ever taken a clock apart to see how it works? (Nerd answer: yes.)
- Did you get it put back together again? (Non-intuitive nerd answer: no.)
- If not, do you consider the experiment a failure? (Nerd answer: no.)

He that breaks a thing to find out what it is has left the path of wisdom.—J.R.R. Tolkien

Here's another test for nerdiness:

You're a nerd if, when you hear, "The wonder of the dancing bear is not how well it dances, but that it dances at all," you think,



"I bet I could teach a bear to dance."

Cringely calls you "stinking gods among men," which seems just rude (and at the same time kind of cool). Po Bronson (him again) says you will try very hard to maintain the image that you care very little about your image.

Nerd Society

So you recognize yourself in these descriptions and you hate having to deal with people and you'll join the cause to reverse the socialization of software, right? What? You say no?

I am Jack's complete lack of surprise.

Because the final irony in the socialization of software is that the making of software is itself a social activity.

It sure seems like there was a time when it wasn't. I cling to my romantic notions of the lone hacker carefully milling the corners smooth on each bit of the first word processing program for microcomputers, or the first such program for the IBM PC, or that killer debugger for the first Macintosh, or any of those other actual demented nerds who did great things in glorious isolation because nobody wanted to hang out with them.

But that's not how it works today. It's frameworks and Wikis and Open Source repositories and giving back to the community and even pair programming. How un-nerdy.

Software development is social, but what passes for social interaction among nerds? According to Paul Graham, one consequence of the limited socialization of nerds is that great hackers clump. They go where the other great hackers go. With the result that "at any given time there are only about ten to twenty places where hackers want to work." Somebody tell me whether that's a bug or a feature.

Trying to make sense of nerd society, I reflect on social insects. Bees, ants. Drones and worker bees. The language developer as queen bee? Maybe not. But it's interesting to think of the flame wars in the Rails community as colony collapse disorder. I'm just thinking aloud, you understand.

Or language communities. Each language succeeds or fails as it develops a community of users. Each language also reflects the personality of its creator. The edifice of ideas at the heart of Lisp was constructed by an autocratic Scotsman, the bricks of Ada were laid by faceless bureaucrats, and the two languages developed different user communities. To me, it's a good thing if, say, Guido's personality keeps the Python community from getting too large. Keeps it from straying from its core values and degrading into Java, I say.

Would Rebol be a household name if Carl Sassenrath got out of his house more? If Adele Goldberg took herself more seriously or Bertrand Meyer took himself less seriously, would Smalltalk or Eiffel rule the OO world?

If nerds have to be social, I guess I'm glad that they are at least as dysfunctional about it as everybody else. Because it seems to me that none of us, nerd or user, has the social thing figured out.

But what do I know.

About the Author

John Shade was born in Montreux, Switzerland in 1962. Subsequent internment in a series of obscure institutions failed to enlighten him so much as a foot-candle. He is a professional skeptic, and currently is directing his most vitriolic skepticism at LA electro-pop band Fol Chen, who promised him a fortune but never delivered.

The Quiz

This Month: If you thought Sudoku was the evil Count's sister…

This month we present a Sudoku puzzle with a twist—it uses letters instead of digits and hidden within it is the name of a well-known person in the development of personal computers.

	A		F	K				
	K		J			A		
							I	N
	N					S		
	I	K						
A			S	F				
N	F			R		J	A	
	J		I				R	
K		R						F

Simply fill in the empty cells with the nine letters A, E, F, I, J, K, N, R, and S, so that each row, column, and small square contains each of the nine letters once.

Answer to Last Month's Quiz

Last month we asked you to identify 6 programming languages and use the first letters of their names to spell a seventh language.

- ```
  Process Class Creator;
  Begin
    While true do begin
      Activate New Consumer(Time);
      Hold(Uniform(5, 15, 1));
    End While;
  End of Creator;
  ```

  This is Simula, the first object-oriented programming language. It's worth a look; but it has some incredible concepts, and some of them still haven't been explored in more modern languages.
  ```

- ```
  <Any T, Any U> T first((T,U) tuple) {
    (T t, U u) = tuple;
    return t;
  }
  ```

  This is an example of Nice [U1] pretending to be Haskell.

- ```
  PROC null.farm(CHAN OF ADDR.TASK.STREAM from.farm,
                 CHAN OF ADDR.RESULT.STREAM to.farm )
     PAR
       from.farm ? CASE no.more.task.packets
       to.farm ! no.more.result.packets
  :
  ```

  This is Occam [U2], a concurrent programming language built around Tony Hoare's idea of Communication Sequential Processes [U3].

- ```
  LET start() = VALOF
  $( FOR i = 1 TO 5 DO writef("%n! = %i4*n", i, fact(i))
     RESULTIS 0
   $)
   AND fact(n) = n=0 -> 1, n*fact(n-1)
  ```

  BCPL [U4], an ancestor of C.

- ```
  PROCEDURE speak*( VAR bird : Birds.Bird );
  BEGIN
      WITH bird : Cuckoos.Cuckoo DO
                  bird.sound := "Cuckoo!";
         | bird : Ducks.Duck DO
                  bird.sound := "Quack!";
      ELSE        bird.sound := "Tweet!";
      END;
  END setSound;
  ```

  Oberon-2 [U5] was created by Niklaus Wirth as an evolution of Pascal and Modula-2.

- ```
  HAI
  CAN HAS STDIO?
  IM IN YR LOOP UPPIN YR NUM TIL BOTHSAEM NUM AN 10
      VISIBLE SUM OF NUM AN 1
  IM OUTTA YR LOOP
  KTHXBYE
  ```

  LOLCODE [U6], the programming languages of our lords and masters.

Simula, Neat, Occam, BCPL, Oberon, and LOLCODE spell SNOBOL, one of the first (and funkiest) text processing languages. Put a state machine into a blender along with a tablespoon of both Perl and Awk, along with a handful of some special mushrooms. Enjoy!

External resources referenced in this article:

[U1] http://nice.sourceforge.net/

[U2] http://en.wikipedia.org/wiki/Occam_(programming_language)

[U3] http://en.wikipedia.org/wiki/Communicating_Sequential_Processes

[U4] http://en.wikipedia.org/wiki/BCPL

[U5] http://en.wikipedia.org/wiki/Oberon_and_Oberon-2_(programming_language)

[U6] http://lolcode.com/about
  ```

Calendar

Aug 3-5	**The 3rd International Workshop on Symbolic-Numeric Computation** [U1] Kyoto, Japan
Aug 3-7	**Siggraph** [U2] New Orleans, LA
Aug 4-7	**iPhone SDK Studio** [U3] Reston, VA
Aug 5	**Ruby on Real Estate: How to Apply Agile Development Principles to Your Project** Mike Mangino, author of Developing Facebook Platform Applications with Rails [U4] Inman ConnectTech, San Francisco, CA
Aug 10-11	**International Computing Education Research Workshop** [U5] Berkeley, CA
Aug 11	**Steve Wozniak's birthday**
Aug 11	**Tom Kilburn's birthday**
Aug 12	**28th birthday of the IBM PC**
Aug 12-13	**OpenSource World: formerly LinuxWorld, collocated with CloudWorld** [U6] San Francisco, CA
Aug 13	**Career 2.0** Jared Richardson, author of Ship It! [U7] NEJug, Boston MA
Aug 13-16	**Filemaker Developer Conference 2009** [U8] San Francisco, CA
Aug 17-21	**ACM SIGCOMM 2009** [U9] Barcelona, Spain
Aug 19	**Gordon Bell's birthday**
Aug 19-21	**Ruby on Rails Studio** [U10] Denver, CO
Aug 19-21	**International Symposium on Low Power Electronics and Design** [U11] San Francisco, CA
Aug 19-21	**Test Automation (Java)** Jared Richardson Chantilly VA
Aug 20	**Build Trust: Keeping People Happy Without Paying an Arm or a Leg (or any other body parts!)** Johanna Rothman, author of Behind Closed Doors: Secrets of Great Management [U12] Biz Conference [U13], Amelia Island FL
Aug 20	**Beyond the Organization Chart: What Really Drives Behavior at Work** Esther Derby, author of Behind Closed Doors: Secrets of Great Management [U14] Biz Conference [U15], Amelia Island, FL
Aug 23-28	**SHARE** [U16] Denver, CO

Aug 24 **Workshop: What Does an Agile Coach Do?**
Liz Sedley and Rachel Davies, authors of Agile Coaching [U17]
Agile 2009 [U18], Chicago, IL

Aug 24 **TDD Clinic: C++**
James Grenning (with Bas Vodde)
Agile 2009 [U19], Chicago, IL

Aug 24 **Increase Your Capacity and Finish Projects: Manage the Project Portfolio**
Esther Derby, Johanna Hoffman
Agile 2009 [U20], Chicago, IL

Aug 24 **Idea Factory**
Brian Marick, author of Everyday Scripting With Ruby [U21]; Programming Cocoa with Ruby [U22], with David Carlton
Agile 2009 [U23], Chicago, IL

Aug 24 **Eight Guiding Values**
Brian Marick
Agile 2009 [U24], Chicago, IL

Mon Aug 24 **Agile Coaching workshop**
Rachel Davies and Liz Sedley
Agile 2009 [U25], Chicago, IL

Aug 24-28 **Programming with the Stars**
James Grenning
Agile 2009 [U26], Chicago, IL

Aug 24-28 **ESEC/FSE 2009** [U27]
Amsterdam, The Netherlands

Aug 25 **Top Ten Tips for Agile Coaches**
Rachel Davies and Liz Sedley
Agile 2009 [U28], Chicago, IL

Aug 25 **Continuous Testing Evolved**
Ben Rady and Rod Coffin
Agile 2009 [U29], Chicago, IL

Aug 25 **Linux turns 18. Free krill for all!**

Aug 25-28 **iPhone SDK Studio** [U30]
Denver, CO

Aug 26 **Workshop: Telling Your Stories: Why Stories are important for your team**
Johanna Hunt and Rachel Davies
Agile 2009 [U31], Chicago, IL

Aug 26-27 **Moving to Ruby 1.9, Module Magic**
James Edward Gray II, author of TextMate: Power Editing for the Mac [U32]
Lonestar Ruby Conference [U33], Austin TX

Aug 27 **Test Driven Development in Java: Live and Uncensored**
Ben Rady
Agile 2009 [U34], Chicago, IL

Aug 27 **Back to Basics—Writing Expressive Tests Without All The Wizardry**
Rod Coffin
Agile 2009 [U35], Chicago, IL

Aug 27 **Speaking**
Dave Thomas
Lonestar Ruby Conference [U36], Austin TX

Aug 27-29 **2009 8th International Conference on Grid and Cooperative Computing (GCC 2009)** [U37]
Lanzhou, China

Aug 28-29 **Testing**
David Chelimsky, author of The RSpec book [U38]
Ruby Hoedown, Nashville TN

Aug 28-30 **Speaking**
Scott Davis, author of Groovy Recipes: Greasing the Wheels of Java [U39]
No Fluff Just Stuff Software Symposium [U40], Research Triangle Park NC

Aug 28-30 **Speaking**
Stuart Halloway, author of Programming CLojure [U41]
No Fluff Just Stuff Software Symposium [U42], Research Triangle Park NC

Aug 28-30 **Speaking**
Jared Richardson
No Fluff Just Stuff Software Symposium [U43], Research Triangle Park NC

Aug 28-30 **Speaking**
Venkat Subramaniam, coauthor of the 2007 Jolt Productivity award-winning book Practices of an Agile Developer [U44]
No Fluff Just Stuff Software Symposium [U45], Research Triangle Park NC

Aug 29 **Jacksonville Code Camp** [U46]
Jacksonville, FL

Aug 29-30 **WOWODC East** [U47]
Montreal, Canada

Aug 30 **John Mauchley's birthday**

Aug 31-Sept 2 **The 18th ACM SIGPLAN Conference on Functional Programming** [U48]
Edinburgh, Scotland

Aug 31-Sept 3 **VMWorld** [U49]
San Francisco, CA

Aug 31-Sept 3 **Chip on the Dunes** [U50]
Natal, Brazil

Sept 2 **Manage the Project Portfolio**
Johanna Rothman
PMI Silicon Valley

Sept 3 **Haskell Symposium** [U51]
Edinburgh, Scotland

Sept 4 **John McCarthy's birthday**

Sept 4-7 **2009 13th IEEE International Symposium on Wearable Computers (ISWC)** [U52]
Linz, Austria

Sept 7 **David Packard's birthday**

Sept 8 **Study of Nanoscience and Emerging Technologies Nanoethics Symposium 2009** [U53]
Seattle, WA

Sept 9 **Dennis Ritchie's birthday**

Sept 9 **64 years ago today Grace Hopper got the Harvard Mark II running again after extracting a moth from a relay, the first computer bug.**

Sep 9 **Pragmatic Thinking and Learning**
Andy Hunt
Trinug, Raleigh NC

Sept 9-10 **Gov 2.0 Summit** [U54]
Washington, DC

Sep 10-11 **Speaking**
Esther Derby, Andy Hunt, Jared Richardson, and others
RubyRX/AgileRX [U55], Reston VA

Sep 10-11 **Speaking**
Rachel Davies
Agile Open Holland [U56], Baarn, The Netherlands

Sept 10-12 **2009 3rd Conference on Affective Computing (ACII 2009)** [U57]
Amsterdam, The Netherlands

Sep 11-12 **Speaking**
Dave Thomas
Agile China, Beijing

Sept 12 **Behaviour Driven Rails with RSpec and Cucumber**
David Chelimsky
WindyCityRails [U58], Chicago IL

Sept 15-18 **4G World** [U59]
Chicago, IL

Sep 19 **Utah Code Camp** [U60]
South Jordan, UT

Sep 19 **Raleigh Code Camp** [U61]
Raleigh, NC

Sept 20-25 **MobiCom 2009** [U62]
Beijing, China

Sept 21-24 **ESC** [U63]
Boston, MA

Sept 22-24 **EmTech09** [U64]
Cambridge, MA

Sept 22-24 **Intel Developer Forum** [U65]
San Francisco, CA

Sept 23 **Speaking**
Johanna Rothman, author of Manage It! Your Guide to Modern, Pragmatic Project Management; Behind Closed Doors: Secrets of Great Management
Agile Boston User Group [U66], Boston MA

Sep 23-27 **Getting Git and Micro-Frameworks in PHP**
Travis Swicegood, author of Pragmatic Version Control Using Git [U67]
Code Workz, San Francisco(23rd), Los Angeles (25th), Dallas (27th)

Sep 25-26 **Speaking**
Jared Richardson
Western Canada Software Symposium, Calgary CA

Sep 26 **Introduction to Test Driven Development and Stuck in Agility?**
Ben Rady
Houston Tech Fest, Houston TX

Sep 27 **Speaking**
Marcus S. Zarra, author of Core Data [U68]
360 iDev Conference, Denver CO

Sept 28-30 **2009 IEEE-PES/IAS Conference on Sustainable Alternative Energy (SAE)** [U69]
Valencia, Spain

Sept 28-30 **NanoTech Europe** [U70]
Berlin

Sept 29-30 **Software Business 2009** [U71]
San Diego, CA

Sept 30-Oct 3 **11th International Conference on Ubiquitous Computing** [U72]
Orlando, FL

Oct 3-4 **Silicon Valley Code Camp** [U73]
Los Altos Hills, CA

Oct 4 **John Atanasoff's birthday**

Oct 4-6 **Behaviour-driven development**
Dan North, author of The RSpec Book [U74]
JAOO, Denmark

Oct 4-7 **Adobe MAX** [U75]
Los Angeles, CA

Oct 4-7 **VSLive** [U76]
Orlando, FL

Oct 5 **Season Opener: South Carolina v. North Carolina**
Supreme Court chambers, Washington, DC

Oct 5-7 **SIGDOC 2009** [U77]
Bloomington, IN

Oct 7-9 **Interop Mumbai** [U78]
Mumbai, India

Oct 11-15 **Oracle OpenWorld** [U79]
San Francisco, CA

Oct 11-16 **International Conference on Hardware-Software Codesign and System Synthesis** [U80]
Grenoble, France

Oct 12 **Pragmatic Thinking and Learning**
Andy Hunt
Skills Matter, London

Oct 13 **Arthur Burks' birthday**

Oct 13-14 **Speaking**
David Chelimsky
Rails Summit Latin America, Sao Paulo, Brazil

Oct 15-16 **International Symposium on Empirical Software Engineering and Measurement** [U81]
Lake Buena Vista, FL

Oct 19-21 **Advanced Ruby Studio** [U82]
Denver, CO

Oct 19-22 **Getting Git**
Dave Klein
Groovy/Grails Experience, New Orleans LA

Oct 19-22 **ZendCon** [U83]
San Jose, CA

Oct 19-23 **Software Test & Performance Conference** [U84]
Cambridge, MA

Oct 19-24 **ACM Multimedia 2009** [U85]
Beijing, China

Oct 20-22 **Web 2.0 Summit** [U86]
San Francisco, CA

Oct 20-22 **Getting Git**
Travis Swicegood
ZendCon, San Jose CA

Oct 21 **Pragmatic Thinking and Learning**
Andy Hunt
JUG, Richmond VA

Oct 21-23 **ARM techcon3** [U87]
Santa Clara, CA

Oct 22 **Pragmatic Lessons Learned in Project Management**
Johanna Rothman
Compaid webinar

Oct 22-23 **HotNets-VIII** [U88]
New York, New York

Oct 22-24 **SIGITE 2009** [U89]
Fairfaz, VA

Oct 23 **Strange Loop Conference** [U90]
University City, MO

Oct 24 **Meet Ruby**
Brian Hogan, author of Web Design for Developers [U91]
The Twin Cities Code Camp, Minneapolis MN

Oct 24 **No More Excuses: Test Your Javascript!**
Kevin Gisi
The Twin Cities Code Camp, Minneapolis MN

Oct 25-28 **Speaking**
Johanna Rothman
PNSQC, Portland OR

Oct 25-29 **OOPSLA** [U92]
Orlando, FL

Oct 26-29 **OCEANS 2009** [U93]
Biloxi, MS

Oct 28 **Ted Hoff's birthday**

Nov 9-13 **Pragmatic Thinking and Learning**
Andy Hunt
Agile Development Practices, Orlando FL

External resources referenced in this article:

- http://www.snc2009.cs.ehime-u.ac.jp/
- http://www.siggraph.org/s2009/
- http://www.pragprog.com/titles/mmfacer/
- http://icer.cs.berkeley.edu/
- http://www.linuxworldexpo.com/
- http://www.pragprog.com/titles/prj/
- http://www.filemaker.com/developers/devcon/index.html
- http://conferences.sigcomm.org/sigcomm/2009/
- http://www.islped.org/
- http://www.pragprog.com/titles/rdbcd
- http://www.bizconf.org/
- http://www.share.org/
- http://www.pragprog.com/titles/sdcoach/
- http://www.pragprog.com/titles/bmsft/
- http://www.pragprog.com/titles/bmrc/
- http://www.esec-fse-2009.ewi.tudelft.nl/
- http://www.pragprog.com/titles/textmate/
- http://agile2009.agilealliance.org/
- http://lonestarrubyconf.com/
- http://grid.lzu.edu.cn/gcc2009/
- http://assets0.pragprog.com/titles/sdgrvr/
- http://assets0.pragprog.com/titles/shcloj/
- http://assets0.pragprog.com/titles/pad/
- http://www.nofluffjuststuff.com/
- http://www.jaxcodecamp.com
- http://www.wocommunity.org/wowodc09/
- http://www.cs.nott.ac.uk/~gmh/icfp09.html
- http://www.vmworld.com/community/conferences/2009/
- http://www.lasic.ufrn.br/chiponthedunes2009/
- http://www.haskell.org/haskell-symposium/2009/
- http://www.iswc.net/
- http://www.foresight.org/cms/events/304
- http://www.gov2summit.com/

- http://www.nfjsone.com/conference/washington_dc/2009/09/
- http://www.agileopen.net/agile-open-holland-2009
- http://www.acii2009.nl/
- http://windycityrails.org/
- http://4gworld.com/
- http://utcodecamp.com
- http://www.codecamp.org
- http://www.sigmobile.org/mobicom/2009/
- http://esc-boston.techinsightsevents.com/
- http://www.technologyreview.com/emtech/
- http://www.intel.com/idf/
- http://www.newtechusa.com/agileboston/
- http://www.pragprog.com/titles/tsgit/
- http://www.pragprog.com/titles/mzcd/
- http://www.cfp.upv.es/SAE/
- http://www.nanotech.net/
- http://www.softwarebusinessonline.com/conf2009/SB/sb_conf09_index.php
- http://www.ubicomp.org/
- http://www.siliconvalley-codecamp.com
- http://www.pragprog.com/titles/achbd/
- http://max.adobe.com/
- http://vslive.com/
- http://www.sigdoc.org/2009/index.html
- http://www.interop.com/mumbai/
- http://www.oracle.com/us/openworld/
- http://www.codes-isss.org/
- http://www.csc2.ncsu.edu/conferences/esem/
- http://pragmaticstudio.com/
- http://www.zendcon.com/
- http://www.stpcon.com/
- http://www.acmmm09.org/
- http://www.web2summit.com/web2009
- http://www.armtechcon3.com/2009/
- http://conferences.sigcomm.org/hotnets/2009/
- http://www.sigite.org/
- http://thestrangeloop.com/Strange%20Loop/Welcome.html
- http://www.pragprog.com/titles/bhgwad/
- http://www.oopsla.org/oopsla2009/
- http://www.oceans09mtsieeeBILOXI.org/