The Pragmatic Bookshelf

PragPub

The Second Iteration

IN THIS ISSUE

Issue #74
August 2015

Contents

DEPARTMENTS

Published monthly in PDF, mobi, and epub formats by The Pragmatic Programmers, LLC, Dallas, TX, and Raleigh, NC. E-Mail webmaster@swaine.com. 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

On Tap

Mob Rule

You can download this issue at any time and as often as you like in any or all of our three formats: pdf [U1], mobi [U2], or epub [U3]. You can also download our special issue on teaching kids to code in all three formats: pdf [U4], mobi [U5], or (you guessed it) epub [U6]. If you subscribed through an app and want to also get the pdf, mobi, and epub file download links emailed to you every month, just ask [U7].

Mob Programming

Woody Zuill knows a lot about Mob Programming [U8]. He characterizes it as "all the brilliant people working at the same time, in the same space, at the same computer, on the same thing." If you're not familiar with the technique, it probably sounds extreme. In fact, if you're comfortable with Extreme Programming [U9], Mob Programming might *still* sound extreme. The whole team working together on one task on one computer? How could that possibly be efficient?

Woody is going to tell you how, in detail, in this month's *PragPub*. And he's going to convince you to give it a try. Just watch.

This month's *PragPub* also features the second installment of our series on teaching kids to code, written by Jim Bonang and filled with insight and case-study experience. Jim found himself thrown into an intense teaching situation, and drew inspiration and guidance from a variety of sources, including a special issue of *PragPub* on teaching kids to code.

If you think Modern Perl is an oxymoron, you haven't been keeping up with the developments in this Swiss Army knife of a language. The fourth edition of the classic *Modern Perl* is coming out soon, and its author, chromatic, took time out from wrapping it up to write an article for us on what Perl *still* gets right. Spoiler alert: it gets *a lot* right.

And there's more. This month's installment in our series on functional programming in Swift focuses on wrapper types. Rothman and Lester often write about how to get hired for the right job, but this month they flip the script and discuss how to write a job ad that will get you the right candidates. Marcus Blankenship also challenges expectations as he explains why managing by walking around is no substitute for old-school weekly meetings. And John Shade has some thoughts on ad blockers. And of course there's our monthly puzzle, a mashup of a Sudoku and an anagram, as well as Antonio Cangiano's list of new tech books. We hope you enjoy the issue.

Who Did What

photo credits: Cover: Vertical Scrum [U10] by Steve Jurvetson is licensed under https://creativecommons.org/licenses/by/2.0/ [U11]. Page 14: Database Plan [U12] by tech_estromberg is licensed under https://creativecommons.org/licenses/by/2.0/ [U13]. Page 22: Mob Mind Map [U14] by Improve it is licensed under https://creativecommons.org/licenses/by/2.0/ [U15]. Page 28: Pearl Has Landed [U16] by Mihalorel is licensed under https://creativecommons.org/licenses/by/2.0/ [U17]. Page 33: chimney swift [U18] by Ed Schipul is licensed under https://creativecommons.org/licenses/by/2.0/ [U19].

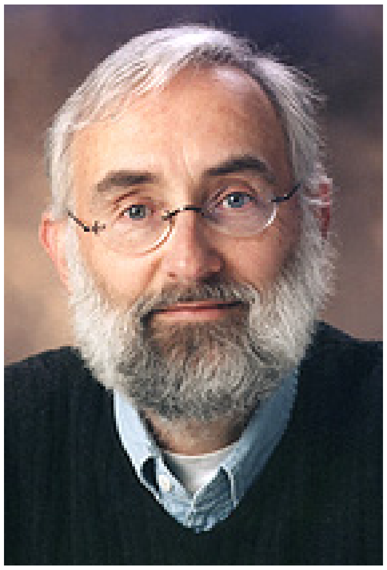
editing and production: Michael Swaine • editing and research: Nancy Groth • customer support, subscriptions, submissions: webmaster@swaine.com • staff curmudgeon: John Shade (john.shade@swaine.com)

Swaine's World

Belly Up to the Bar

by Michael Swaine

Some choice bits to nosh on.



"It never rains in a pub." — Traditional Irish saying

Here are a few choice bits to nosh on. Tweets. A bill of fairly interesting sites. Our Pub Quiz. Or sudoku/anagram mashup, really. A List. That sort of thing. Cheers.

Agile Angst

- *Apparently I need to write something about: Agile Software Development is a kind of Software Development and REQUIRES Software Development.* — @RonJeffries
- *Do we think that /Excessively Asynchronous Systems/ overdoes the sibilance? Would /Outrageously Asynchronous Systems/ be better?* — @marick
- *Perhaps instead of ScrumMaster we should start calling them ScrumServants.* — @danielbmarkham
- *"I've decided I'm going to measure how my most important measurements fail to measure the most important things about life." #trap #paradox* — @berkun
- *If you open a meeting with a multi-minute monologue, you should have sent an email instead.* — @therealadam
- *You think email doesn't scale? I agree. It doesn't. But you know what scales exponentially worse? Chat.* — @codinghorror

Amazon's Fire Sale

- *Amazon Prime Day surpassed Black Friday in rate of sales.* — @LaurenGoode
- *When I die, I want whoever was responsible for #PrimeDay to lower me down into my grave so I can be let down one more time.* — @simonzhow
- *If Amazon had a sense of humor, they'd sell their remaining stock of Fire Phones as IPS LCD luggage tags for $5 a throw today.* — @lhnatko
- *In honor of #PrimeDay a #library near you will be lending books for FREE! Fun fact: They do this on other days as well.* — @libshipwreck

I'm a Fool Stack Developer

- *When did full stack developer start to mean "JavaScript all the way down"?* — @bphogan
- *Every frontend developer is super stoked about another browser to support.* — @harrisj
- *Showing React.js to a PHP developer must be what it felt like to show fire to a caveman for the first time.* — @iamdevloper
- *Definitely the post I've been most looking forward to writing all year: self-hosted ClojureScript is REAL.* — @swannodette
- *Derp as a Service.* — @xxx

The Pub Quiz

Here's this month's brain-teaser.

	Y				U			
		S						X
X					N	A		
A		Y				X		L
		X					S	O
N							U	
			X	L				S
			Y		A	O		
	I							

It's a sudoku, but with letters. Using only the nine letters that appear in the grid, fill in the empty cells so that every row, every column, and each of the nine 3x3 boxes contains all nine letters.

And it's also an anagram. Properly arranged, the nine letters in the puzzle will reveal the secret word. Clue: it's how you might solve this puzzle (although I don't recommend it).

Solution further on in the issue; don't peek!

Internet of the Thing from Another World

- *Hacked jeep reportedly ab* — @jwgrenning
- *8-Year-old boy gets Lego-compatible prosthetic arm.* — @GuyKawasaki
- *Disembodied mechanical random clickerfinger.* — @doctorow
- *I'd take autonomous AI controlled weapons over internetworked weapons. @elonmusk @stevewoz @ID_AA_Carmack* — @sneakin
- *Yep, even after sleeping on it for a night: Someone please remake @StephenKing's Christine as a self-driving car malware movie.* — @acroll
- *And thus the aliens came and took the internet away, claiming we were not ready to handle the immense powers of memetic teledildonics.* — @sargoth

The Egg and I

- *Early morning gym sessions are my time to think through hard problems like "what's my name again?" and "why did I set my alarm so early?"* — @seldo
- *Ever since I put all my eggs in one basket I have received unsolicited egg advice. You don't know my life, you don't know what I'm all about.* — @wolfpupy

Pub Crawl

This month's pub crawl hits a bunch of food-related museums. I hope you won't ask me why. Hat tip to This Belongs In a Museum [U1], which must be the biggest collection of links to odd museums ever. It's kind of an odd museum of odd museums.

Museums house the hard to find, the easy to forget, the rare and the exemplary. The Baked Bean Museum of Excellence [U2] and the Sardine Museum & Herring Hall of Fame [U3] presumably display particularly noteworthy beans and fish.

There are museums devoted to bananas [U4], onions [U5], currywurst [U6], and chewing gum [U7]. There are museums catering to your craving for the lore of asparagus [U8], ramen [U9], pickles [U10], salami and paprika [U11]. The lowly potato merits museums in Idaho [U12], Canada [U13], and Germany [U14].

And who knew that Taiwan was famous for its nougat [U15] or China for watermelons [U16]? Taiwan's Museum of Drinking Water [U17] sounds ominous.

But we can't let you go thirsty on a pub crawl. So if you don't drink the water on your museum travels, just ask for a sample at Moscow's Vodka Museum [U18], Cuba's Rum Museum [U19], Vienna's Brandy Schnapps Museum [U20], or the World of Coca-Cola Museum [U21], Dr Pepper Museum [U22], or Illinois' Root Beer Saloon [U23].

The ultimate food museum pub crawl, of course, would have to include Minnesota's Spam Museum [U24], New York's Jell-O Museum [U25], and Nebraska's Kool-Aid: Discover the Dream [U26]. Bon appétit!

Pluto

- *TIL: I have a serious Pluto crush.* — @pragpub
- *Lovely! Pluto has a heart on it! https://instagram.com/p/5HTXKMoaFL/* — @timoreilly
- *On the left is the best image we had of Pluto before. Now we have this. Just astonishing. #PlutoFlyby pic.twitter.com/rA1Ane3oVt* — @kayaburgess
- *From Hubble to New Horizons. Like me getting up, then putting my specs on. pic.twitter.com/UemAcEpSkp* — @brucel
- *Can I go work *way* remotely now and do so in orbit around Earth, please?* — @lindseybieda

Information Superhighway Road Rage

- *We are unanimous in Our disdain for Them. They ruin everything for Us.* — @pragpub
- *We don't police our conduct to avoid "offending" anyone. We do it to avoid hurting each other, and our communities by driving others away.* — @peeja
- *"The Internet launders outrage and returns it to us as validation, in the form of likes and stars and hearts."* — @mims
- *If you feel you will make the world a better place by being angry at a dentist, why not support http://www.lionconservationfund.org/join.html instead (or as well)?* — @neilhimself

The Writing Is Off the Wall

- *The New York Times is adding digital subscribers at nearly double the pace (5%) of Twitter's user growth (2.6%). $TWTR $NYT* — @dkberman

- *My current favorite writing advice is: sleep on it. You don't really know what you need to fix until you haven't thought about it for a bit.* — @rands
- *Regular PSA: "Ask": Still not a noun.* — @chadfowler
- *Resonance remains my favourite word in the universe.* — @xxx
- *Things programmers know how to fix for sure according to my Twitter feed: Greece's economy. Things broken for the past 50 years: software.* — @monsieur_pickle
- *If I'm reading this recent funding data right, FOMO is the new YOLO.* — @StartupLJackson

The Punters

Here are the folks I followed this month: Jeff Atwood, Scott Berkun, Dennis K. Berman, Lindsey Bieda, Andrei Bocan, Ed Bott, Kaya Burgess, Alistair Croll, Cory Doctorow, Chad Fowler, Neil Gaiman, Gay as heck, Lauren Goode, James Grenning, Jacob Harris, Brian P. Hogan, I Am Devloper, Andy Ihnatko, Startup L. Jackson, Peter Jaros, Ron Jeffries, Guy Kawasaki, Adam Keys, Bruce Lawson, LibrarianShipwreck, Brian Marick, Daniel Markham, Christopher Mims, Nolan, David Nolen, Tim O'Reilly, rands, Michael Swaine, Laurie Voss, David Weinberger, Jo White, wolf pupy, and Simon Zhou. But fair's fair. You can follow me at www.twitter.com/pragpub [U27]. Or visit my blog at swaine.com [U28].

- *Best Facebook post of 2015: Somebody: Need to bring back Bloom County! Berkeley Breathed: Watch this space.* — @pragpub

Rothman and Lester

Selling the Job

by Johanna Rothman and Andy Lester

Managing your career is a job in itself. Fortunately, Johanna and Andy have seen all the career mistakes people make and can steer you past the hidden rocks. This month's lesson focuses on job descriptions.

Andy: When you're looking to hire someone for your team, the job ad is the first, and sometimes only, impression the potential candidate will get to have about your company and the position you're trying to fill. In some companies, HR always writes the job ads, but if you're fortunate enough to work for a company that lets the department write their own ads, it's worth putting quite a bit of thought into what you're going to say, and how you're going to say it. You've got to look at it from the perspective of an outsider looking in.

Johanna: If you read job ads, you see things such as, "Back-end developer," or "UI/UX developer," or "Java developer." You might need a developer you hire to do any one of those things. And, starting a job description like that is boring and doesn't offer an opportunity to a person. You have other choices for how you define your open developer job.

Andy: The job description in the ad is not the same as the job description that has been written up by your HR department. That is likely to be extremely vague and full of legalese. It's certainly not going to entice anyone to apply for the job you have open.

Johanna: If you want to offer someone an opportunity, think about what would make a job enticing for *you*. Would you like to know about the domain, the team, or the organization's mission? Each of those is part of the job analysis [U1]. Add in the issue of cultural fit, and now you have an opportunity for someone.

Andy: I like to start with brainstorming a laundry list of relevant technologies. Don't skip any because they're "obvious." If you're in a Vim shop and you use a certain distro of Linux, then go ahead and put that down. You may not need it, but it could be relevant in later writing stages. Get everything that is part of your day-to-day work life. You might wind up with a list like "Apache, MySQL, PHP, Perl, Red Hat Enterprise Linux, Vim, Nagios, Photoshop, Mac OS X." If there are any specific technologies, like certain Perl or Ruby modules you use extensively, include them as well. Get as much dumped as you can.

Johanna: I like to start with who the person will work with and the non-technical skills. I start this way because candidates may have the same technical skills, and it's how the person works that matters. The organization might make this job exciting. It's the same idea as what Andy suggests — you are trying to find the whole picture of the job.

Andy: Next, do the same thing with the work activities. Braindump everything this person might need to do. Your list might start out with: "Maintain the Foo web app; maintain the company wiki; keep servers X, Y, and Z running; run monthly reports about A, B, and C for upper management; create ad-hoc data analysis solutions for the Marketing department; etc., etc., etc." Err on the side of too much rather than too little.

Johanna: When you describe the deliverables you expect, you are itemizing some of the problems you want that person to solve. You and I might both be able to solve the problems — *how* we do it is what sets us apart.

Do you need someone who can collaborate across the organization, in a team, or both? Do you need someone with the ability to mentor others? These non-technical skills are what set candidates apart from each other.

Andy: You'll create a couple of useful lists that will come in handy during the interview process. Then, it's time to take those lists and start creating a job ad.

Johanna: I offer templates [U2] for *Hiring Geeks That Fit* for a job analysis (what we have described so far) and the ad. Once you've analyzed the job, everything is much easier.

Describe an opportunity in the ad. Explain some of the deliverables and some of the technical and non-technical qualifications. Your ad should help people screen themselves in if they would fit the role, or out of the position if they don't meet the qualifications.

Andy: I always make sure that there are three parts to the job ad: Information about the company, about the technical skills required, and about the work to be done. They don't need to be in any certain order, but they need to all be there.

For information about the company, tell about what the company does with some references to why it's a good place to work.

For technical skills required, go back to your laundry list of skills you made earlier and pull out the most important and the most distinctive.

For the work to be done, give a bit of an indication of what it's like to work in your shop, and why someone would want to work there.

For example, you could come up with something like this:

Yoyodyne Inc., the world's No. 1 maker of gravel-sifters, is looking for a web developer with expertise in Nginx, Ruby, and PostgreSQL to maintain both internal and B2B web apps. You'll work on a small, focused team of five in our modern developer-friendly office.

Johanna: When you write an ad like this, you explain not only what the job is in terms of technical skills, but also you help people understand the culture they can expect. When I read Andy's example, I know there are internal and external customers. The fact that there are five people on the team means we probably collaborate a lot to finish the work. A "modern developer-friendly office" means I probably don't have to fight for a build machine.

Here's an ad that emphasizes different things:

Yoyodyne, a Fortune 200 organization, seeks a developer with the flexibility to work back-end through front-end on its agile team for the flagship product, YoYoProduct. You'll work with the product owner to define the features. You'll work with a collaborative team to release every day. We offer tuition reimbursement and an internal conference each year, where we encourage our people to learn from one another.

This ad emphasizes the environment you would work in. This organization is looking for a "generalizing specialist" type of developer, which is why it's light on the specifics for the technology.

Both ads emphasize the offer, not just the technology or the toolset.

Andy: Back in the old days when we sent want ads to get printed in the newspaper, we were charged by the word, so we wrote only what was absolutely necessary. Nowadays, publishing ads online, we have more room to say more of what we want to say. However, just because you can include lots of text in your ad doesn't mean you should. You want your ad to get read, and tons of text that isn't relevant or enticing isn't going to get the reader to read through to the end. Say what you need to say, then stop.

Finally, ask candidates for feedback about how your ad worked for them. It's perfectly OK to ask a candidate in the interview what attracted her to the job, or what caught her eye in the ad. You can also ask if she felt the ad was missing information that would have helped her before the interview.

Johanna: Let me recap our tips. Offer a candidate an opportunity. Define your opportunity by thinking about the entire role, not just the technical skills you require. Write an ad that describes what's unique about your company and this position, so candidates can screen themselves in or out.

Once you have a candidate you like, ask that person what appealed to them about the ad. You now have feedback to do an even better job the next time.

About the Authors

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. She writes the *Pragmatic Manager* email newsletter and two blogs on www.jrothman.com [U3].

Andy Lester has developed software for more than twenty years in the business world and on the Web in the open source community. Years of sifting through résumés, interviewing unprepared candidates, and even some unwise career choices of his own spurred him to write his nontraditional book Land The Tech Job You Love [U4] on the new guidelines for tech job hunting. Andy is an active member of the open source community, and lives in the Chicago area. He blogs at petdance.com [U5], tweets at @petdance, and can be reached by email at andy@petdance.com.

External resources referenced in this article:

[U1] https://pragprog.com/book/jrgeeks/hiring-geeks-that-fit

[U2] http://www.jrothman.com/wp-content/uploads/2013/03/HiringGeeksThatFitTemplates.pdf

[U3] http://www.jrothman.com

[U4] http://www.pragprog.com/refer/pragpub51/book/algh/land-the-tech-job-you-love

[U5] http://petdance.com

New Manager’s Playbook

The Case for Weekly Meetings: Why an Old-School Schedule Gets Leading-Edge Results

by Marcus Blankenship

You’re a developer, but you’re at a point in your career where you find yourself managing others. Marcus shares tips on how to be as good at managing as you are at your “real” job.

This month: why “management by walking around” may need to take a hike.

In these days of open office arrangements, Agile project management, and a flood of fast-and-light startup practices, the idea of sit-down weekly meetings seems almost quaint, like chalkboards and rotary phones.

Even Hewlett-Packard abandoned the practice of weekly meetings years ago in favor of “management by walking around.” Managers were encouraged to get out of their corner offices and to meet their team members at their cubicles, right in the midst of their work.

While this and other philosophies seem very proactive or egalitarian in their approach, I’ve found that the good-old weekly meeting is the best communication tool I have in my management toolbox, even with a small agency where everyone works in one small room or as remote freelancers.

Weekly meetings aren’t flashy management hacks, but they have saved me from countless hours of frustration, piles of revisions, and team conflicts.

Week in and week out, I hold onto my weekly one-on-one meetings, and I make sure I keep them with employees, customers, and my own boss.

Here’s why.

The Hidden Benefits of Weekly Meetings

If music is what happens between the notes, then much of management is what happens around the meetings.

In other words, some of the power of a weekly meeting comes from the discussion inside that set time and location, but just as much power comes from your dedication to the meeting.

Here are some of the unspoken benefits you’ll get from an unwavering commitment to weekly meetings.

You’re signaling to your team and your company that you are dependable.

Many small organizations have meetings that have no real fixed time or agenda. This is done in the name of flexibility (as in, “Hey, whenever it works for you — just let me know.”), which sounds great. But it gives the impression that meetings aren’t very important, so they often are overlooked and then cancelled.

Insisting on timely, fixed meetings signals not only your discipline and consistency, but also your commitment to your own employees and their input.

Remember that as a manager, you are a communication hub [U1], and these meetings are transmission lines. Without them, everyone is left running on assumptions.

Hands down, the weekly meeting is the best way to have a (nearly) no-surprises workflow.

When you host and hold onto weekly meetings, you'll hear far more from your employees than if you waited for them to initiate conversations with you.

In this safe, supportive space, you can set priorities and keep tabs on each employee and project *without looking over their shoulders each day.*

But most important, you'll have the chance to make gentle corrections so you can catch project potholes before they create full-fledged sinkholes. I can't tell you how many times weekly meetings have provided a place for me to straighten out a misunderstanding that would have cost us dozens of billable hours.

Other times, I've been able to nudge a team member in the right direction without a whiff of a reprimand. Since everyone cringes at the thought of difficult conversations [U2], this communication tool is a perfect, painless way to offer guidance — sometimes, without the employee even knowing they're being corrected.

How to Get Maximum Productivity from Weekly Meetings

Not all meetings are created equal, however. If you're used to meetings being stumbling, drawn-out affairs that steal time from true productivity, let me give you a few pointers on how you can make meetings *work* for you.

Sticking to these principles has given me a lot of management cred *without* having to lay down the law because they set unwritten but clear standards for communication.

- *Keep the same time, length, and location each week.* Consistency commands respect and provides stability in an industry that requires exceptional adaptability.
- *Have the meeting in a separate location, like a conference room or a coffee shop every time.* This usually allows for privacy and gives an edge of formality and focus to the discussion. If your programmers are freelancers, schedule an online meeting with them.
- *If your employee misses a meeting, do not go fetch them.* It's not your responsibility to make sure they remember the meeting.
- *The moment you notice a team member no-shows, wait 10 or 15 minutes, then send them a cancel/reschedule notice.*
- *When you do meet again, make sure you ask, "What happened?" as an open-ended question.* Don't provide an out for them — let them answer in their own words. You might be surprised to hear the legitimate answer.

Now that we have the foundation in place, let's build a strong structure on top of it.

The 5-Point Weekly Meeting Agenda

If you thought the idea of a weekly meeting was old-school, you'll probably really cringe at my next suggestion.

After years of tinkering with format and content, I've settled on the conclusion that the best way to organize these meetings is to use each employee's timesheet.

Yes, I said "timesheet."

1) Anchor meetings around the timesheet

A timesheet contains so much information about an employee's workflow that would take me forever to collect, so I ask each team member to bring their own timesheet and a short, pre-written status report for discussion (bullet points are just fine).

Even if timekeeping is done electronically, I ask employees to print out a copy *before* the meeting (or have freelancers email it to me) so we don't waste time printing it or spend time looking at a screen instead of interacting with each other.

2) Focus on accomplishments, not accounting

New employees will probably be a bit nervous about bringing in their timesheet because it feels initially like bringing a report card to the principal. But after a few trial runs, they'll soon understand that this is a feedback loop, not a called-to-the-carpet moment.

The point of this meeting is truly to take a minute to shine a light on all of the successes that usually are forgotten in the rush of the next item on the TODO list. In other words, it's a time that your employees get to brag a bit.

3) Review where the time was spent

Start the meeting with the open-ended statement: "Please take me through your week."

When you review each project and how much time was given to each, you get a rapid, clear sense of a few key issues:

- Your team member's skill level
- Understanding (or misunderstanding) of priorities
- Problem areas that need attention
- Unexpected victories
- Missed details

Keep an eye out for places they're getting stuck, effective strategies they've found for saving time or eliminating problems, and places where they need to follow up to finish the job correctly.

4) Look ahead

After you've reviewed the previous week, ask them to set three priorities for the coming week — and write those down. Making sure everyone understands the plan of action eliminates wasted time and frustration.

Since this point is so important, I have the employee write these priorities on their timesheet, which I file and use to review the following week to see if the priorities and time spent are the same or different.

If we find that the priorities didn't match the time spent, there may be a good reason why, so don't leave the employee feeling like the priorities are unchangeable or the documentation will be used against them later. The priorities are just guideposts which can be adjusted as needed.

Also, take a few minutes to ask if the employee has any scheduled out-of-office days (i.e., vacation, professional development) planned for the next couple of weeks. Keeping an eye on time off can help the entire team plan their workflow and deadlines.

5) Set up a follow-up meeting if needed

Yes, this is a lot of information to cover in one meeting, so if you hit an issue that needs more unpacking, set another meeting to discuss just that issue.

Keeping a disciplined 30-minute structure to meetings helps maintain focus. It's very easy to get sidetracked for legitimate reasons, and a strict time limit teaches everyone to keep rolling so production can move forward.

If they have a question for you that requires some research, like asking upper management, don't just say, "I don't know, but I'll find out." Make sure you completely understand *why* they have a question so you can get an accurate answer.

Communication and Momentum

Once I realized that my job as an owner-manager was primarily about communication — and I made communication a priority — I've seen my team's productivity jump significantly, sometimes by 25 to 30 percent!

I say this to reassure you as you experiment with weekly meetings and add up the precious time spent checking in with your team.

While there will always be unexpected glitches and delays, the weekly meeting has been my best defense against needing the firehoses or lifeboats on a regular basis.

About the Author

Nearly 20 years ago I made the leap from senior-level hacker to full-on tech lead. Practically overnight I went from writing code to being in charge of actual human beings.

Without training or guidance, I suddenly had to deliver entire products on time and under budget, hit huge company goals, and do it all with a smile on my face. I share what I've learned here in *PragPub* and here [U3].

External resources referenced in this article:

[U1] http://marcusblankenship.com/chapter-1-lead-from-the-podium-6-essential-concepts-new-technical-agency-owners-miss/

[U2] http://marcusblankenship.com/chapter-11-defusing-frustrating-situations-with-gentle-corrections/

[U3] http://marcusblankenship.com

Making Computer Science Insanely Great

Part 2: Insanely Great Presentations

by James Bonang

Jim learns the presentation techniques of a master while teaching kids to code, and explains why presentation skills are essential for software professionals like you too.

This is Part 2 of a 3-part story of one developer's experience in teaching kids to code.

Sometimes great ideas just stare you in the face. It feels a bit odd though, when they literally stare at you.

I had wanted to introduce computer science at my daughter's high school but before I could put my plans in motion, the U.S. Navy recalled me to active duty and deployed me overseas. One leg of the deployment brought me to a naval base with a small high school serving the children of U.S. service personnel. The school happened to be planning a Science, Technology, Engineering, and Math (STEM) activities night and I submitted my application as a volunteer instructor.

As Emerson suspected, the universe does indeed conspire to make your decisions happen: I was selected to teach the computer science activity. I would have thirty minutes to introduce my students to programming, to give them a useful skill and a sense of accomplishment. And in the process make computer science exciting and interesting. No pressure.

I selected a topic and developed a lesson plan that fit within the time and equipment constraints with the help of the high school's remarkable teachers and the authors of the *PragPub* "Teaching Kids to Code" series of articles. My students would create Vokis [U1], customized, animated speaking characters, embed them in their own web pages they would write in HTML, and have the characters debate the pros and cons of school uniforms. (All recounted in Part I in the July issue of *PragPub*.)

But how do you present technology, programming, and computer science in a way that makes it exciting and interesting — to high school students? How do you arrange a presentation to accomplish that? Who knows how to do that? A question I pondered as I sat staring at the blinking cursor within Pages, the word processing program I used to compose my notes on my recently acquired Apple MacBook Pro. I happened to glance up at the little Apple logo in the Pages menu bar, then my eyes darted across my small desk and I stared at the cover of a new autobiography, recently arrived from home via military post. The subject of the book stared back at me: "Who knows how to do that? That guy does!"

Insanely Great Presentations

Steve Jobs [U2] knew about getting people interested in technology (Isaacson, Walter. Steve Jobs, New York, Simon & Schuster, 2011, ISBN 978-1451648539). In his 2007 MacWorld keynote address, Jobs introduced his electrified audience to the iPhone. Watch the video [U3]; after the applause subsides, Steve demonstrates the iPhone's innovative multi-touch user interface. But the world hasn't seen anything quite like this before; it's not so much a marketing demonstration as a lesson — he's teaching. And his audience is mesmerized. It's like magic. How did he do that? Could his presentation techniques be applied to teaching students computer science? Could I learn them?

Steve's keynote presentations helped propel Apple to new records, eventually becoming the most valuable U.S. company in 2015. His techniques merit study in detail and journalist Carmine Gallo [U4] did just that, and summarized his observations in *The Presentation Secrets of Steve Jobs: How to be Insanely Great in Front of Any Audience* [U5] (McGraw-Hill Education, 2009, ISBN 978-0071636087). A few of Steve's techniques are specific to product marketing, but you can apply most of them to teaching technical topics.

Of Steve's techniques, one stands out as absolutely essential. Fail at this and nothing else matters; handle this first technique with care.

Why Should I Care?

Have you ever sat in class and wondered "what use is this?" Your audience will, and you need to answer the question "why should I care?" near the very start of your presentation. Your answer should be concise, specific, convey a personal benefit and be repeated a few times.

When he introduced the iPod, Jobs answered with "One thousand songs in your pocket." And he would repeat his answers to "why should I care?" at least twice during his presentations. His answer is concise, like a newspaper headline or Twitter tweet. A five- or six-word newspaper headline may persuade you to read an article. A tweet (140 characters) might persuade you to read an author's blog.

I answered the question verbally while on my title slide: you'll acquire an immediately useful skill. And within the first few slides I also pointed out other benefits. Avatars can be used to spice up class presentations and you'll "earn better grades." I emphasized this later with a few slides that used embedded Voki avatars to describe the contribution of computer scientists such as Rear Admiral Grace Hopper [U6]: a Voki of Admiral Hopper describes her career. Once the students understood that Vokis are customized, animated speaking characters, I pointed out they could "have fun" by emailing Vokis to friends.

So, when you present (no matter what the topic, no matter who the audience), answer the question "why should I care?" with a headline-length (or tweet-length) phrase that conveys a tangible, specific, personal benefit, and repeat it. Jobs did. Every time.

Steve made sure you cared about what he was talking about. But even so, after all these years, why can so many recall his presentations so vividly?

Modeling Memorable Moments

Unlike MacWorld audiences, students need role models. For my role model I selected an actual model: supermodel Lyndsey Scott [U7]. When she's not on the runway, Ms. Scott, who earned a degree in computer science from Amherst College, develops applications for the iPad such as a portfolio management tool for artists. As a supermodel, she doesn't need the money; she programs for fun. She likes to create things. Ms. Scott's example implies all sorts of benefits: "programming is fun," it enables you to "express your creativity," "the cool kids do this and you can too," and "programming can be a desirable occupation."

More importantly, none of the students saw this coming — a supermodel who programs? What? When my teenage daughter reviewed the presentation, she commented "I can't believe that a supermodel develops software!" Using Lyndsey Scott as a role model built a memorable moment into the presentation — a technique right out of Steve's playbook. A Steve Jobs keynote includes a memorable moment: remember when Steve pulled that MacBook Air from the manila envelope? Of course you do — Steve planned it that way. Include a "memorable moment" in your presentations; your audience won't forget it.

Steve explained why you should care and made sure you wouldn't forget his latest technological marvel, but he placed these within an overarching pattern, a pattern that would be easily recognizable to Tolkien or Homer.

That's My Story and I'm Sticking to It

Pulitzer Prize-winning author Jon Franklin defines a story as:

. . . a sequence of actions that occur when a sympathetic character encounters a complicating situation that he confronts and solves: *Writing for Story: Craft Secrets of Dramatic Nonfiction* by a two-time Pulitzer Prize Winner [U8]. (Plume, 1994, ISBN 978-0452272958, p.71.)

Steve Jobs organized his presentations this same way, just like a story: he would state a problem and present its solution, raise a rhetorical question and provide an answer, introduce an antagonist and vanquish them. Complication and resolution. Timeless. Most famously, Jobs portrayed IBM as Orwell's Big Brother from *1984* [U9]: Levy, Stephen. *Insanely Great: The Life and Times of Macintosh, The Computer that Changed Everything* [U10]. (Kindle eBook, p. 181), an antagonist whose Personal Computer would be vanquished by the Macintosh, which Jobs stealthily unveiled [U11] from a small, unobtrusive carry bag placed beside him on stage at the 1984 Apple shareholders meeting.

Here, I stumbled. I didn't see the importance of casting the lesson as a story. I introduced a halfhearted problem — what to do with a Voki avatar once you've created one. My solution was to incorporate it into a web page, and have two Vokis conduct a debate. And, of course, this fell flat. When you teach, when you present, follow Steve's example, make it a story. Introduce a complication and provide a resolution. It's worked pretty well for the past few millennia.

Of course, some stories can be hard to follow. But not Steve's.

Make It Easy

Why is it so easy to follow a Steve Jobs presentation? Why doesn't your mind wander? How did Steve make it so easy on his audience? Two ways.

First, he limited the content in his presentation to three (or at most four) major items, staying within the capacity of short term memory [U12]. He never overwhelmed his audience. He began his famous commencement address [U13] at Stanford University by saying "Today, I want to tell you three stories from my life."

He also chunked subtopics into groups of three or four, facilitating retention. For example, in 2005 he described the iPod as incorporating three major breakthroughs: "ultra-portable," "built-in Firewire," and "extraordinary battery life." I stuck to Steve's "Rule of Three" in my presentation. I included three major topics: 1) How to create a Voki Avatar, 2) How to write an HTML page, and 3) how to incorporate a Voki into a webpage. Make it easy on your audience: limit your content to three (or four) major items. "Chunk" subtopics into groups of three too. It just works.

Steve used a second technique to make it easy on his audience: he would include a context-switch at ten-minute intervals. After ten minutes, our brains get bored; we start looking at the clock. Steve would insert a video, bring up a new speaker, or give a demonstration to break things up. He'd provide that intermission our brains need, right at the ten-minute mark. Like clockwork.

My lesson included plenty of breaks: Voki videos would be used to explain key points. The students would perform hands-on exercises to build a Voki, or create an HTML file, after receiving "just-in-time" instruction. I easily kept to the ten-minute rule. But there's a corollary to the rule that I violated. *Badly*. Video clips used to break up your presentation should be kept short: two or three minutes at most. I included a clip showing how to create a Voki — it lasted six minutes. *Disaster*. So, include a context-switch after at most ten minutes. But beware the video clip: no more than two or three minutes. Make it easy on your audience — it's just that easy.

Now, a story that's easy to follow isn't necessarily exciting. However, when you recall Steve Jobs, *dull* is not the first word that comes to mind.

A Man of Few (Written) Words

Cognitive researcher John Medina of the University of Washington determined that the average PowerPoint slide contains forty words. 40. Four zero. The first four slides of the Steve Jobs MacWorld 2008 keynote contained seven words (along with three numbers and a date) — minimalist, as though crafted by designer Deiter Rams [U14]. The words spoken in accompaniment to the slides were a different matter. Awesome. Revolutionary. Great. Extraordinary. Incredible. Amazing. Steve used all of these words on the first slide. The slide had four characters on it: 2 0 0 7. Steve conveyed his excitement with abandon.

For my presentation, I used words like fun, cool, and awesome to convey my excitement about the topic. Professionals, particularly programmers, tend to be a bit subdued, understated. (Does anyone remember Ben Stein in Ferris Bueller's Day Off? [U15] Anyone? Anyone?) High school students, in contrast, view understatement as the scenes between explosions in the Avengers movies.

Steve kept the word count down on his slides but the excitement up in his speech. When you teach, convey your excitement. You're up against superheroes after all. When you present to professionals, adjust your vocabulary yet still convey excitement. But convey that excitement in the words you speak and keep the words on your slides to a minimum.

While I used exciting words while teaching my lesson, I also used a lot of words on my slides. An awful lot of words. Way too many words. I made an egregious mistake. *Epic fail!* Worse, I didn't realize it. Paid no attention. Software engineers conduct retrospectives to review what went right, and what went wrong, on their projects. My slide construction debacle will get a retrospective all its own.

Steve made his presentations exciting, easy to follow, memorable stories. And he made sure you knew why his topic mattered. Now, making all that happen doesn't happen overnight.

Nothing Prepares You Like Preparation

Steve's preparation is legendary. He would begin weeks in advance of a presentation, reviewing the technologies and products he would discuss. Former Apple employee Mike Evangelist, a member of Steve's preparation team, reports in his article "Behind the Magic Curtain" [U16] in the January 5th, 2006, edition of *The Guardian* that he and his team spent hundreds of hours preparing for a five-minute demonstration of (then novel) software for burning DVDs. The behind-the-scenes effort is astounding: making it look easy *isn't* easy. (They provide all sorts of backup systems!) For your presentations and lessons, plan on at least ninety hours of preparation for a one-hour presentation. My lesson was half an hour and I spent forty-seven hours preparing. It wasn't enough.

Um, Ah, You Know

Not practicing a presentation is like not testing new software. Not a good idea. I should have known better.

Your audience gives you a substantial amount of their time: they deserve excellence. When Steve presented, he gave nothing less. How did his presentations come off so polished and apparently effortless? With lots of effort. With weeks of practice. He maintains eye contact with the audience. His gestures come at just the right moment. There's nary an "Um" or "Ah" in his speech.

The U.S. Navy trains its personnel to present briefings. They take people, often without any intrinsic talent (like me), and turn them into adequate briefers. Not into Steve Jobs mind you, but still not bad briefers. One technique they use right at the start left me stunned. I really couldn't believe it, just could not believe it, but what they told me was completely accurate. You would present your briefing, typically for just a few minutes, and, among other things, a panel of three reviewers would count your Ah's, Um's, and You Know's. Usually, they all agreed on the number to within one or two Um's. I came in at twenty three! You don't notice it, but, ah, you do it. You really don't realize it. But your audience does. Ah's and Um's are filler words. Useless words. *Annoying* words. How do you exorcise filler words from your speech?

Practice.

In part, filler words arise from nervousness. And you defeat nervousness with practice. Try recording yourself during a practice run — just the first five minutes will suffice. A camcorder would be best, but your computer will do nicely too. Then count the filler words. Once you become conscious of the Um's, you'll catch yourself and eliminate them within a few iterations. You'll become more confident too; your nervousness will evaporate. But like me, listening to that recording the first time might leave you stunned.

My preparation trouble didn't involve eliminating filler words though; I needed my *own* Mike Evangelist.

Practice Like You Preach

STEM night loomed large on my calendar: time was running out. The school invited all the volunteer instructors in about a week prior to review their materials and to allow them to inspect the venue. Take advantage of this when offered. The teachers reviewed my draft presentation and pronounced it sound, but were worried it wouldn't be finished in time. Then they showed us the computer lab. The lab staff didn't allow anything to be placed on the school computer systems. I couldn't plug in a USB or insert a DVD. I could connect my computer to the audio/video display though, but I discovered I needed an adapter. I'd need to purchase that. I wasn't familiar with my new Macintosh — I'd never used OS X before. After building the prototype of the presentation, I was at least minimally familiar with Keynote and Pages (the Macintosh's presentation and word-processing applications, respectively), but that was about it. I asked to come in an hour before the lesson to set up my equipment and troubleshoot if necessary; they agreed.

Practicing your presentation is like testing and debugging your software; you work out the unanticipated glitches. I plugged my Macintosh into the lab's overhead speakers and played one of my videos. Silence. Nothing. I knew audio output worked on the Mac and assumed (incorrectly) that there was a problem with the lab's speakers. I'd just borrow a pair of speakers I could plug into the Mac instead. I didn't know that if you plug a speaker in, any adjustments to the volume previously made to the internal speakers didn't apply to other audio devices; the external speaker volume remained at the default — zero. I should have tested this in advance. Didn't practice. *Big mistake*.

The armed forces train as realistically as is feasible, they "train like they fight." When you present, "practice like you preach." Practice in a setting as close to the actual presentation as possible. Debug your presentation. I didn't learn this from Steve — I learned this the hard way.

Intermission

At this point, I had a lesson plan that conformed to the guidance from the authors of the *PragPub* articles on teaching kids to code. I also had a solid draft of the presentation, though not a finished one, and I emulated Steve's techniques with a fair amount of success.

The table below summarizes how well I did following Steve's techniques:

Technique: How I Did

Answer the question: "Why should I care?"	Check. Get better grades, have fun, emulate glamorous supermodels, get a good job.

Include a memorable moment.	Check. Supermodel Lyndsey Scott as a role model.
Make your presentation a story: introduce a complication and provide a resolution.	Poor. Complication: what to do with a Voki. Resolution: use it on a web page.
Limit your content to three (or four) major items; chunk subtopics into groups of three.	Check. Three major topics: (1) Build Voki (2) Build webpage (3) Insert Voki in webpage.
Switch context after at most ten minutes; provide an intermission.	Check. Videos and hands-on kept the presentation segments to well under ten minutes each.
Keep video clips to at most two or three minutes.	Fail. One video was six minutes, but all others were less than one minute.
Convey your excitement by using "zippy" words like extraordinary, incredible, and amazing.	Absolutely!
Keep the words on your slides to a minimum.	Fail. Slides contained far too many words.
Plan on at least ninety hours of preparation for a one hour presentation.	Check. Forty-seven hours spent for a half hour presentation; wasn't enough.
Practice. Record a practice session and eliminate filler words.	Uh (strike that) Check. However, didn't practice enough and didn't "practice like you preach."

Table 1: Steve Jobs' Presentation Techniques

I emulated all but two of Steve's techniques: I couldn't check off "keep the words to a minimum on the slides"; my slides were much too verbose. (We'll see a technique to solve this problem soon.) One of my videos was too long. Still, I viewed my progress with some satisfaction. These are the most important of Steve's techniques and the most apropos to teaching, and I abided by nearly all of them.

More importantly, you can now employ the techniques of a legendary presenter in your lessons, or when presenting to your software development team, to management, or to those venture capitalists. Of course, Steve used many more techniques, and you'll find them all in Carmine Gallo's bestseller *The Presentation Secrets of Steve Jobs: How to be Insanely Great in Front of Any Audience* [U17].

While I adhered to most of Steve's guidance, I fell short on following my own advice: "practice like you preach"; practice in a setting as close to the actual presentation as possible and debug it.

As a programmer I should have known better. Nonetheless, I could see the finish line now. I still needed to craft step-by-step instructions and edit my slides, but how hard could that be? Almost there! I would be teaching at last! As it turned out though, I still had much to learn.

…which will be revealed next month.

We've created a special issue of PragPub containing all of our articles on teaching kids to code that are referenced in this article. You can download the issue in any or all of our three formats here: pdf [U18], *mobi* [U19], *epub* [U20].

About the Author

Jim Bonang has spent over twenty-five years developing software for a wide range of systems, including satellite terminals, document management systems, medical devices, and many others. He's also worked on several compilers.

External resources referenced in this article:

[U1]	http://www.voki.com
[U2]	http://www.amazon.com/Steve-Jobs-Walter-Isaacson/dp/1451648537/
[U3]	https://www.youtube.com/watch?v=wGoM_wVrwng
[U4]	http://carminegallo.com
[U5]	http://www.amazon.com/The-Presentation-Secrets-Steve-Jobs/dp/1491514310
[U6]	http://en.wikipedia.org/wiki/Grace_Hopper
[U7]	http://www.forbes.com/sites/vannale/2014/08/18/meet-lyndsey-scott-model-actress-and-app-developer/
[U8]	http://www.amazon.com/Writing-Story-Dramatic-Nonfiction-Reference/dp/0452272955/
[U9]	http://www.amazon.com/1984-Signet-Classics-George-Orwell/dp/0451524934/
[U10]	http://www.amazon.com/Insanely-Great-Macintosh-Computer-Everything-ebook/dp/
[U11]	https://www.youtube.com/watch?v=RcRQWGFJ5YY
[U12]	http://en.wikipedia.org/wiki/Memory
[U13]	http://news.stanford.edu/news/ 2005/june15/jobs-061505.html
[U14]	http://en.wikipedia.org/wiki/Dieter_Rams#Rams.27_ten_principles_of_.22good_design.22
[U15]	https://www.youtube.com/watch?v=uhiCFdWeQfA
[U16]	http://www.theguardian.com/technology/2006/jan/05/newmedia.media1
[U17]	http://www.amazon.com/The-Presentation-Secrets-Steve-Jobs/dp/1491514310
[U18]	https://s3-us-west-2.amazonaws.com/pp-x1/Special.pdf
[U19]	https://s3-us-west-2.amazonaws.com/pp-x1/Special.mobi
[U20]	https://s3-us-west-2.amazonaws.com/pp-x1/Special.epub

Mob Programming

United We Code

by Woody Zuill

Because Extreme Programming isn't extreme enough.

Mob programming is a software development approach in which the whole team works on the same thing, at the same time, in the same space, and at the same computer. Similar to pair programming where two people collaborate with one keyboard, Mob Programming extends the collaboration to everyone on the team.

The Beginning

In an effort to increase our Agile knowledge and development skills, we were spending two to three hours each Friday studying Extreme Programming techniques such as pair programming, refactoring, and test-driven development. We adopted a Coding Dojo style where the entire team joined together to work through simple code exercises. With one computer, one projector screen, one keyboard, and one mouse, the whole team worked together to solve one code problem. Studying together and frequently reflecting about our progress provided a hidden benefit: We were learning how to work as a team.

The teamwork skills we were building really paid off for us one day when we took on a fairly large project that had been put on hold a few months earlier. We gathered together and opened the project files to start looking at it together. We naturally started using the teamwork techniques we had practiced in our weekly Coding Dojos. Each person would contribute just the right thing at just the right time, and almost every question we had was quickly answered by another team member.

At the end of that first day we all agreed we were getting a lot done, and decided to continue working together in the same way the next day. After a week of working this way we found we were becoming very effective at working together to quickly turn out high-quality code and useful features so we decided to find a permanent work area and continue working as a "mob" full time. We haven't stopped since — and that was more than four years ago!

I Know What You're Thinking

How could this possibly work?

While it might seem that it would be more productive to have five people work on separate tasks than having those same five people work together on a single task, in practice we found this isn't always true. We discovered that some of the problems that typically plague development efforts simply faded away once we became good at Mob Programming, and a lot of good things were happening.

For example:

- With all of us working on the same thing together, the best of each of us goes into everything we do. Where I am weak, another is strong.
- By working on one thing at a time, an idea goes from inception to "in use" in just a few hours. Each product grows daily so we can validate both the value and the correctness of everything we do as fast as possible.
- We never put off cleaning up the code, so very little technical debt enters our code. When working alone it's easier to think "I'll clean that up later," but with the group someone will say "we better clean this up right now." Almost nothing shady sneaks into the code.
- Most questions we come up with as we work can be quickly answered by another member of the team. We have very little blocking and context-switching due to unanswered questions.
- We engage our internal customers as team members: They come and sit with us daily as we work for an hour or two. This provides shared understanding and rapid, meaningful communication, and results in features that are delivered into production that do what the users want. We get quick validation of the usefulness of the code.
- As a team, we have enough familiarity with anything we are working on so that when someone is out sick or on vacation there is no blocking due to lost knowledge.
- There is a very shallow learning curve for a new team member to go from joining the team to being able to contribute meaningfully.
- Everyone is learning and increasing their skills continuously. We are all exposed to what everyone else knows, and anytime we find something we don't know we all learn it as a team. There are no information or skill silos.

There is a great deal of overhead in managing the typical team. We noticed most of that overhead disappeared when we started "mobbing." What if working together as a team actually eliminates the cost and work of managing software development? I think it has for us to a great degree.

The Basic Principles and Practices

On a Mob Programming team, everyone works together to do almost all the work a typical software development team tackles, such as defining stories, designing, coding, testing, deploying software, and working with the customer or product owners (who we also consider to be part of the team).

To make it possible to work together full-time requires a rethinking of how we typically set things up for "solo" work. For one thing, it's necessary to reconfigure the physical workspace to accommodate a team, and even more important, we need to find ways to make it easy to get along. We strive to accentuate and amplify concepts such as face-to-face and side-by-side communication, team alignment, collaboration, sustainability, whole team involvement, continuous code review, and self-organizing around the work at hand. Finding practices and techniques that allow us to interact well has paid off nicely for us.

While we have no rules and follow a very dynamic approach to our daily work, we do have a few basic principles and a few simple practices we use to make this collaboration go smoothly. I share these here as an example, and you are welcome to use them if you wish, but I encourage you to craft your own approach to uniquely address your team members and workplace situation.

Principles

Principle: *Honor "Individuals and Interactions" over just about everything else.*

The first value listed in the Agile Manifesto [U1] makes it clear how important it is to find ways to make it easy for individuals to:

- Get work done
- Contribute to the effort
- Work on the right thing and
- Work well with others

There are plenty of great practices available, but not all serve us as well as we would hope. We adopted the "Individuals and Interactions" Agile value as a way to judge any practice we thought might be useful to us. I'll share a bit more about this under the practices section below.

Principle: *Treat each other with kindness, consideration, and respect.*

Because we work closely together throughout the day, we are constantly communicating and interacting, and we must be able to express ideas, discuss problems, explore possible solutions, resolve conflicts, and share thoughts all day long.

At first we simply depended on the ability and desire of each individual to get along well. However, it didn't take us long to realize we needed a formal agreement on how we would behave toward each other. We crafted a simple principle to follow, which is this: "We will always strive to treat each other with kindness, consideration, and respect."

This seems straightforward, yet expressly acknowledging the importance of this principle provided a foundation for our daily interactions, which has served us well. It's amazing how quickly we can improve when we are all dedicated to this simple principle. I've noticed for myself that by working with a group of people who have all agreed to help each other follow this principle I have become a much better person than I had been. This one idea alone might be worth trying even if you're not able to do Mob Programming in any substantial way.

Practices

Practice: *The Driver/Navigators Model*

An emphasis on the "Individuals and Interactions" Agile value becomes critical when 5 or 6 people are working on the same thing together. It is of utmost importance to make it easy for everyone to know what's going on so they can pitch in when they can be of help.

The basic guideline is this:

"For an idea to go from your head into the computer, it must go through someone else's hands."

Following this approach requires that we get good at communicating our ideas verbally and at the whiteboard, and ensures that everyone has a chance to discuss and scrutinize everything we do. It exposes all ideas to the entire team, and provides continuous design review.

During our coding dojos, we had learned the Driver/Navigator practice. This pattern of pairing was shown to me by Llewellyn Falco [U2] a number of years ago, and has been named "Strong-style Pairing" by Arlo Belshee [U3]. As you might have guessed, there are two roles we use: the Driver and the Navigator.

The driver sits at the keyboard and types in the code. The navigators (everyone else on the team) express and discuss the problem, propose solutions, and the idea to be coded. We switch up the driver frequently, and everyone who wants to take the keyboard has ample opportunity to do so.

Working together, the navigators guide the driver in creating the code. The navigators are expressing their ideas to the driver and each other in a metered approach so the driver only has to focus on the next thing to code at any given time. The driver has a much more mechanical job than when coding solo. The driver listens to the navigators and focuses on translating ideas into code, asking questions as needed.

As we discuss and work out the possibilities, both verbally and at the whiteboard, everyone is gaining a full understanding of the current task and solution. This creates a sort of collective intelligence across the team. We generally speak at the highest level of abstraction that the driver is able to digest at the moment. This can be at a very high level when the driver understands the concept to be coded and can proceed without detailed instructions, but it can also be at a very detailed level, even at the level of keystroke instructions when needed. This will change from person to person and idea to idea.

Practice: *Timed Rotation*

We use a timer and a randomized rotation pattern. Every morning we create a new rotation list, and each team member works at the keyboard as the driver for a short period of time, typically ten to fifteen minutes, in turn. The current driver hands the keyboard off to the next driver when her turn ends, and rejoins the navigators. We don't require that everyone take the driver role; it is your own choice whether to do so.

Practice: *Whole Team*

Our approach follows the Extreme Programming practice of "whole team," where every contributor to the project is an integral part of the team. We have all the skills we typically need for almost everything we do, including coding, designing, database, testing, business expertise, and documentation. When we don't have the skills or knowledge we need within the team, we find someone who does and invite him or her to work on the team to accomplish the needed work.

For example, testing is done as a team effort. The business people (product owners, end-users, expert users, etc.) join our team as first-class members and we work along side them for any testing that needs to be done. Several of us have extensive testing experience, and two of us have managed testing efforts as well. This is the idea of a true cross-functional team: Someone on the team has the knowledge and skills required, and they can guide the rest of us in doing the work.

Practice: *Reflect, Tune, and Adjust Frequently*

We follow the Agile Principle that “At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.” We have taken this to heart and found it brings a great deal of value to us. We frequently evaluate what is working for us, what problems we might be having, and how we can improve on things. By simply paying attention and then tuning and adjusting, we have been able to choose actions that have led to tremendous improvements for us.

Practice: *Turn Up the Good*

This is the real story of Mob Programming: We noticed that working together was giving us great results and simply decided to do more of it. We “turned it up.” This comes from Kent Beck [U4], a prominent pioneer of Extreme Programming: “When I first articulated XP, I had the mental image of knobs on a control board. Each knob was a practice that, from experience, I knew worked well. I would turn all the knobs up to 10 and see what happened.”

We believe that if we pay attention to what is working well for us, and then find ways to “turn it up,” good things will happen. This is what made it possible for us to discover Mob Programming and many other great things. I am certain it will make it possible for you to discover great things for your team.

A Few Other Odds and Ends

I have noticed that there are a few questions that come up whenever I share Mob Programming with people. Let’s look at a few of them.

How do you deal with conflicts?

There will often be conflicting ideas. That is both expected and welcomed. We are not worried about finding the “one best way” to solve any problem, but instead we’re looking to find one of the many “best” ways. Personal experience and preference will sometimes mean that someone will insist that their solution is best. That’s okay. When conflicting ideas come up, we quickly evaluate them, and steer towards the simpler one. We try it and evaluate the outcome. We can then decide if we still need to try the alternative idea.

When two people insist that their idea is best we don’t need to guess which is better. We’ll try them both and see what we think afterwards. When there is a clear winner we go with that one, and otherwise move forward with one or the other based on a quick polling of the team. After doing this a few times we found there are few situations where we don’t quickly resolve the conflict. No one takes it personally when their idea isn’t chosen, and we all understand that we can reverse any decision if we feel we went down the wrong path.

How many team members is the right number?

This is perhaps the most asked question about Mob Programming. I’m not sure that there is a “right number,” or that it is worth trying to determine that. We use this heuristic: “Do I feel I am contributing sufficiently, or learning something useful working with the team today?” If so, we stay with the team and continue participating, and if not, then it might be a good time to split off and work on something solo or with a pair for a while. If the team needs you, they’ll ask you for help.

We have typically worked with 5 or 6 people in our daily practice. It’s not uncommon for us to have up to 7 or 8 team members, and a few times we’ve had 11 or 12 people actively participating. If everyone involved is providing value, either through contributing or learning, then it’s great.

How do you convince the “boss” that this is good?

I’m not sure that I can help you with this one. One observation: Most people involved in software development are smart and can sense when good things are happening. When we work together in a spirit of teamwork, innovation, and discovery we can get wonderful results. Almost anyone can recognize wonderful. The main challenge is to make it possible for wonderful to happen; to nurture an environment where wonderful is possible.

This quote from Ed Catmull of Pixar makes it clear:

“Protect new ideas from those who don’t understand that for greatness to emerge, there must be phases of not-so-greatness.”

I hope you have a boss that understands that. If not, I encourage you to work on becoming the boss, and then you can bring this idea to your teams. I hope that if you are a boss, you understand and promote this idea.

Conclusion

Mob Programming is the story of how one team learned to pay attention to the things that were working well for them, and to amplify the goodness. A number of other teams at other companies all around the world are trying these ideas and finding similar results. If you decide to try it, I suggest you spend a few learning sessions working on a Code Kata [U5]or solving a problem together until you’ve worked out the kinks. Pay attention to eye strain, backaches, and other health concerns. Make it fun, reduce stress and pressure, and focus on helping each other. You just might find it works for you.

About the Author

Woody Zuill has been programming computers for 30+ years. Over the last 15+ years he has worked as an Extreme Programmer, Agile Coach, Application Development Manager, and Trainer. He believes that code must be simple, clean, and maintainable so that we can realize the Agile promise of Responding to Change, and that we must constantly “Reflect, tune, and adjust.” His passion is for tackling code that is hard to maintain and cleaning, refactoring, and bringing it back into a manageable state.

External resources referenced in this article:

[U1] http://www.agilealliance.org/the-alliance/the-agile-manifesto/

[U2] http://llewellynfalco.blogspot.com/2014/06/llewellyns-strong-style-pairing.html

[U3] http://arlobelshee.com/what-makes-a-good-test-suite/llewellyn-falco/

[U4] http://www.amazon.com/Extreme-Programming-Explained-Embrace-Change/dp/0201616416

[U5] http://codekata.com

What Perl Still Gets Right

An Appreciation of a Venerable Language and Its Rich Ecosystem

by chromatic

Fresh from updating his classic book *Modern Perl*, chromatic explains why Perl is still awesome.

The first time I used Perl, I'd written a proof-of-concept in Bourne shell running on HP-UX 9. The program had to be ported to something more portable — at least to run on Linux and Windows. Java was a plausible option until I looked at its available libraries for sending email. The best at the time was a very thin wrapper over raw SMTP.

I found what I needed on the CPAN and happily wrote a lot of programs in Perl after that.

These days, you can list a dozen great options for doing the same task, and you'll probably leave out several great languages. Whether you like dynamic programming, functional programming, objects, server-side, manifest typing, type inference, compiled, interpreted, JITted, or whatever, you can get from point A to point B with cross-platform languages and libraries, most of them truly free software running pretty much anywhere you want them to run.

A few are already installed for you.

You may never have to touch Perl code in your professional life, but the language's influence on open source languages has been profound (sometimes inspiring an equal and opposite reaction in design). The language that was once dominant for system administration and the first among equals in web programming still has a lot of lessons to teach.

Compatibility and Quality

For the most part, a program written to run in the early '90s on Perl 3 or 4 will run today on the latest stable Perl release, Perl 5.22. With very few exceptions, code written to run on any Perl released since 2000 (38 stable releases of Perl in that time period, eight of them major releases) runs without modification. This includes CPAN modules.

Every well-structured CPAN distribution has a test suite. Whenever you install a distribution, the Perl toolchain runs these tests. Many CPAN developers report their results to a service called CPAN Testers [U1].

CPAN Testers collects these reports and organizes them by platform (OS, architecture, etc.) as well as Perl version. If you upload a new distribution, you can almost watch in real time as you get pass/fail reports with diagnostics. That's really useful for testing your code on platforms you don't necessarily control.

"That's just Continuous Integration," you may say — but it goes further.

Perl itself — the language and the standard libraries — has monthly development releases. It also has a standard test suite that's expected to pass on every platform with every commit. Thanks to the CPAN testers, it has an expanded test suite of millions of lines of real-world code. Some of these testers can and do run the monthly releases against the distributions they care about and report the results.

A few brave souls even put together a project called BBC, for “Bleadperl Breaks CPAN,” which bisect commits to Perl itself to find out exactly which change breaks which modules when something goes wrong. This gives the maintainers of Perl plenty of information about the effect of any change and helps everyone, CPAN authors and Perl maintainers both, fix bugs.

This means that unintentional changes and regressions are avoided. Intentional changes (API changes, changes to undefined behavior) are identified early. Most of these are fixed before the annual major release. If you’ve written your code well, you may even be able to run your code, unmodified, on the newest release of Perl the day it’s released. Just install the new version, install your module dependencies, and go. Additionally, the CPAN dependency service [U2] tracks the dependencies of all CPAN distributions. If you’re choosing between multiple plausible alternatives, you can analyze their dependency chains to see which one best fits your stack.

Maybe that’s not magic to people using compiled languages with runtimes, but if you paid the upgrade cost in the PHP 3 or 4 days or wrestled with Ruby 1.9.x and Rails 2 or left a trailing comma in a list in JavaScript running on IE 6, you never want to go back to that situation again. Sometimes I miss that level of stability when I have to match Gems to Ruby or Rails versions or translate code in my head to the latest Javascript framework of the week.

Usability

Another great feature of the CPAN is often overlooked: documentation. You can go to the CPAN right now and see documentation for just about every module available. For example, the DBIx::Class manual [U3]. You can even download documentation in ePub or Mobi formats. Then again, the documentation gets installed for you when you install CPAN distributions; you can browse it from the command line with the core `perldoc` utility, render it to printable documentation with other modules, or run a local web server and flip back and forth between classes, modules, functions, and distributions the way you’d read documentation online.

Make no mistake, this isn’t “Oh, there’s an example of use and some autogenerated API docs in a `README.md` sometimes, if you can get to the GitHub repo for the gem.” CPAN modules have a well-established standard for documentation, including running code you can often copy and paste and modify into your own programs. Sometimes that’s all you need.

Larger projects and libraries and frameworks, such as Moose [U4] and Mojolicious [U5] have lengthy manuals and guides that explain everything from the very basics to advanced features.

This is a community expectation. It’s part of Perl’s culture. It’s part of the CPAN’s DNA. Once you get used to it, you miss it when it’s not there (ASCIIcasts painstakingly transcribed from video just aren’t the same).

Flexibility

It's easy to praise the CPAN as one of the first major libraries of reusable, freely licensed code. CPAN Testers and meta::cpan and CPAN Deps demonstrate the value of building on this repository (and they're only a few of the many additional services built around the CPAN). There's more to the CPAN, though.

To paraphrase Justice Brandeis of the U.S. Supreme Court, the CPAN is the laboratory of Perl. Language extensions from syntax features to internal improvements are prototyped on the CPAN — sometimes with multiple, competing implementations — to discover the real-world uses, limits, and advantages of each. Some of these improvements have made it into the core. Others, of course, are just as useful and usable as CPAN extensions.

The *Modern Perl* book talks a lot about Moose [U6], a complete object system built from a complete metaprogramming system. It extends Perl's minimal object system to provide features borrowed lovingly and liberally from languages such as Smalltalk and CLOS, all with a Perl-ish spin. Moose transformed the way I write OO code in Perl, allowing me to take advantage of immutable objects, lazily-evaluated attributes, and dependency injection with minimal tooling — and minimal education of coworkers. In one project, we even wrote our own custom metaobject system to manage the work of persisting information to multiple data stores.

Moose is transformational technology, but it's far from the only gem on the CPAN. (Sure, it makes Perl's minimal default object system far more usable, but then it goes far beyond what most other languages have ever provided.)

In a personal project I use an OO extension called Moops [U7]. It adds additional syntax to the Moose ecosystem as well as type constraints and method signatures. Perhaps it's easier to show than tell:

```
use Moops;
class MyApp::Service::Sponsors
  extends MyApp::Service
  types MyApp::Types :ro {
  has 'sponsors_rs', lazy_build => 1;
  method _build_sponsors_rs {
      $self->schema->resultset( 'Sponsor' )
  }
  method list_sponsors {
    $self->sponsors_rs->find_all({ order_by => 'sponsor_id' })
  }
  method find_sponsor_by_id(Int $sponsor_id) {
    $self->sponsors_rs->find_by_sponsor_id( $sponsor_id );
  }
  method update_sponsor(Sponsor $sponsor, HashRef $params) {
    if (my $feed_url = delete $params->{feed_url}) {
      $sponsor->fetchable->update({ url => $feed_url });
    }
    $sponsor->update( $params );
  }
}
```

There's a lot to unpack here. The `class` keyword declares a class which inherits (`extends`) from a parent class. The class also uses type-checking provided both by Moops and by a custom type library for this project. The class also defaults to read-only attributes (:ro). They'll get reader methods, but not writers.

Line five declares one of those attributes, sponsors_rs. This is a ResultSet provided by DBIx::Class; no need to get into the details of the database access layer now. As you might expect, this class has a method called sponsors_rs() which returns the value of this attribute. It's read-only (:ro).

There's a little twist with lazy_build. This is a Moose feature which initializes the attribute once, on demand, on *first* demand. I've used a builder method called _build_sponsors_rs() here. The leading underscore indicates that it's a private method. There are other ways to enforce this, but this is enough encapsulation for my purposes. This method returns the ResultSet alluded to earlier by calling a method on the schema() accessor inherited from the parent MyApp::Service class.

The rest of the code is methods. Note that both find_sponsor_by_id() and update_sponsor() have signatures with types. Moops will enforce that $sponsor_id is an integer, that $params is a reference to a hash, and that $sponsor is an instance of the MyApp::Schema::Result::Sponsor class. (That latter type is defined in MyApp::Types, as you've already intuited.)

Yes, Moops adds type checking to Perl. Although it happens as the program runs — not during up-front compilation — it catches bugs, especially if you've written good tests.

This entire code example is a service class which acts as an adapter between a database-backed model and a front-end component, such as the controller of a web application or a batch job. It has to be simple, direct, and readable. The equivalent code written without Moops (or without Moose) would be much longer, much less clear, and would run more slowly.

Moops isn't your only OO option. Another well-established option is Method::Signatures [U8]. Damian Conway has also just released a distribution called Dios [U9], which looks very interesting. This is a space worth watching, as Perl 5.22 added function signatures to the language and subsequent major releases in 2016 or 2017 may add a slimmed-down version of the Moose metaobject system to the Perl core.

Plenty of other languages have extension mechanisms (CLOS and Smalltalk are two great examples), but the practicality of Perl, the care put into documentation and tooling around the CPAN, and the commitment to quality in the language and its tests all keep me coming back to the language for some of my own projects. I miss them when I'm working in other languages.

To me, that's a compliment to the design of Perl and its ecosystem and its community: it's changed the way I think about programming for the better.

About the Author

In two decades of using Perl, chromatic has written tests, added features, documented hacks, submitted patches, and advocated for quality, consistency, and predictability. He's written code for financial analysis, clinical trials, web forums, and games. Outside of work, you can find him cooking, walking dogs, and playing board games.

External resources referenced in this article:

[U1] http://testers.cpan.org/

[U2] http://deps.cpantesters.org/

[U3] https://metacpan.org/pod/distribution/DBIx-Class/lib/DBIx/Class/Manual.pod

[U4] http://metacpan.org/release/Moose

[U5] http://metacpan.org/release/Mojolicious

[U6] http://moose.perl.org/

[U7] http://metacpan.org/release/Moops

[U8] http://metacpan.org/release/Method-Signatures

[U9] http://metacpan.org/release/Dios

Functional Snippets

Wrapper Types

by Chris Eidhof, Wouter Swierstra, and Florian Kugler

Here's the next installment in our series designed to help you get a grip on functional programming in Swift.

Swift opens up a whole new world of programming. To quote Swift's creator, Chris Lattner: " 'Objective-C without the C' implies something subtractive, but Swift dramatically expands the design space through the introduction of generics and functional programming concepts."

In this series, we use small snippets of Swift code to explore this new world of programming, as well as demonstrate how Swift supports the functional paradigm.

Sometimes you want to distinguish two types from each other. For example, if you have user accounts with credits on them, you could have a function that asks the server for an account's credits:

```
func credits(account: Account) -> Int
```

However, somewhere deeper in your code, you might see this Int and not know that it describes the number of credits. Therefore, it can be very helpful to add a typealias, so you can make the function return a value of type Credits:

```
typealias Credits = Int
func credits(account: Account) -> Credits
```

After doing this you can pass around values of type Credits. This turns out to be very useful when reading code: it is immediately clear what Credits is about. We can even take this one step further: let's assume that Credits come from the backend, and nowhere in our client code do we want to modify it. To prevent us from accidentally treating Credits as an Int, we could wrap it in a struct:

```
struct Credits { let amount: Int }
```

The nice thing about this is that the compiler warns you when you try to do something like Credits(amount: 0) + 1 or even adding two values of Credits to each other. Wrapping a simple integer in a new type can prevent a lot of accidental mistakes. This is also very useful when dealing with things like currencies, physical units, and so on. You can also wrap other types: in many cases it might make sense to wrap a String type once and then use it in multiple places.

About the Authors

Along with Daniel Eggert, Chris Eidhof and Florian Kugler created objc.io [U1], a periodical about best practices and advanced techniques for iOS and OS X development. Eidhof, Kugler, and Wouter Swierstra also wrote the book *Functional Programming in Swift* [U2], in which they explain the concepts behind functional programming and how Swift makes it easy to leverage them in a pragmatic way, in order to write clearer and more expressive code.

Antonio on Books

Thirty New Books

by Antonio Cangiano

Antonio looks at all the new tech books of note.

Antonio Cangiano is the author of the excellent Technical Blogging [U1] and you can subscribe to his reports on new books in technology and other fields here [U2].

When compiling a list of books and an intro paragraph for this short column, I always remind myself that the audience of this magazine is quite technical. As a result, I have a certain bias towards practical, technical books that I'd want to read for myself. This month I'm going to highlight a book that doesn't particularly appeal to me, but that is important. I'm talking about the sixth edition of *Cracking the Coding Interview, 6th Edition: 189 Programming Questions and Solutions*.

These days I'm more on the other side of the table, being the interviewer rather than the interviewed, but I think this is a useful book that can help you in securing a new position as a programmer if you are in the market for one.

Among the books that came out this month, I also found several interesting releases from O'Reilly Media that are worth checking out. Finally, if you are more academically inclined, you might enjoy *The Little Prover*.

Our Staff Pick: *Cracking the Coding Interview, 6th Edition: 189 Programming Questions and Solutions* [U3] • By *Gayle Laakmann McDowell* • ISBN: *0984782850* • Publisher: *CareerCup* • Publication date: *July 1, 2015* • Binding: *Paperback* • Estimated price: *$29.59*

Web Scraping with Python: Collecting Data from the Modern Web [U4] • By *Ryan Mitchell* • ISBN: *1491910291* • Publisher: *O'Reilly Media* • Publication date: *July 24, 2015* • Binding: *Paperback* • Estimated price: *$25.48*

Bioinformatics Data Skills: Reproducible and Robust Research with Open Source Tools [U5] • By *Vince Buffalo* • ISBN: *1449367372* • Publisher: *O'Reilly Media* • Publication date: *July 23, 2015* • Binding: *Paperback* • Estimated price: *$33.13*

Using R With Multivariate Statistics [U6] • By *Randall (Randy) E. (Ernest) Schumacker* • ISBN: *1483377962* • Publisher: *SAGE Publications, Inc* • Publication date: *July 24, 2015* • Binding: *Paperback* • Estimated price: *$38.00*

An Introduction to Machine Learning [U7] • By *Miroslav Kubat* • ISBN: *3319200097* • Publisher: *Springer* • Publication date: *July 18, 2015* • Binding: *Hardcover* • Estimated price: *$47.99*

CSS Secrets: Better Solutions to Everyday Web Design Problems [U8] • By *Lea Verou* • ISBN: *1449372635* • Publisher: *O'Reilly Media* • Publication date: *July 3, 2015* • Binding: *Paperback* • Estimated price: *$21.87*

The Little Prover [U9] • By *Daniel P. Friedman, Carl Eastlund* • ISBN: *0262527952* • Publisher: *The MIT Press* • Publication date: *July 10, 2015* • Binding: *Paperback* • Estimated price: *$30.60*

Docker: Up and Running [U10] • By *Karl Matthias*, *Sean P. Kane* • ISBN: *1491917571* • Publisher: *O'Reilly Media* • Publication date: *July 3, 2015* • Binding: *Paperback* • Estimated price: *$19.50*

Head First Android Development [U11] • By *Dawn Griffiths*, *David Griffiths* • ISBN: *1449362184* • Publisher: *O'Reilly Media* • Publication date: *July 3, 2015* • Binding: *Paperback* • Estimated price: *$27.35*

R For Dummies [U12] • By *Andrie de Vries*, *Joris Meys* • ISBN: *1119055806* • Publisher: *For Dummies* • Publication date: *July 7, 2015* • Binding: *Paperback* • Estimated price: *$16.27*

Minecraft Modding For Kids For Dummies [U13] • By *Sarah Guthals*, *Stephen Foster*, *Lindsey Handley* • ISBN: *1119050049* • Publisher: *For Dummies* • Publication date: *July 13, 2015* • Binding: *Paperback* • Estimated price: *$15.37*

Hadoop Application Architectures [U14] • By *Mark Grover*, *Ted Malaska*, *Jonathan Seidman*, *Gwen Shapira* • ISBN: *1491900083* • Publisher: *O'Reilly Media* • Publication date: *July 20, 2015* • Binding: *Paperback* • Estimated price: *$32.69*

Python for Data Science For Dummies [U15] • By *John Paul Mueller*, *Luca Massaron* • ISBN: *1118844181* • Publisher: *For Dummies* • Publication date: *July 7, 2015* • Binding: *Paperback* • Estimated price: *$17.25*

Raspberry Pi Projects For Dummies [U16] • By *Mike Cook*, *Jonathan Evans*, *Brock Craft* • ISBN: *1118766695* • Publisher: *For Dummies* • Publication date: *July 7, 2015* • Binding: *Paperback* • Estimated price: *$13.76*

Polymorphism: As It Is Played [U17] • By *Joseph Bergin, Ph.D* • ISBN: *1940113059* • Publisher: *Slant Flying Press* • Publication date: *July 20, 2015* • Binding: *Paperback* • Estimated price: *$18.98*

Real-World Kanban: Do Less, Accomplish More with Lean Thinking [U18] • By *Mattias Skarin* • ISBN: *1680500775* • Publisher: *Pragmatic Bookshelf* • Publication date: *July 3, 2015* • Binding: *Paperback* • Estimated price: *$17.90*

Accumulo: Application Development, Table Design, and Best Practices [U19] • By *Aaron Cordova*, *Billie Rinaldi*, *Michael Wall* • ISBN: *1449374182* • Publisher: *O'Reilly Media* • Publication date: *July 20, 2015* • Binding: *Paperback* • Estimated price: *$31.61*

Hadoop Security: Protecting Your Big Data Platform [U20] • By *Ben Spivey*, *Joey Echeverria* • ISBN: *1491900989* • Publisher: *O'Reilly Media* • Publication date: *July 16, 2015* • Binding: *Paperback* • Estimated price: *$32.81*

Beginning C for Arduino, Second Edition: Learn C Programming for the Arduino [U21] • By *Jack Purdum, Ph.D* • ISBN: *1484209419* • Publisher: *Apress* • Publication date: *July 1, 2015* • Binding: *Paperback* • Estimated price: *$33.38*

The Scrumban [R]Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban [U22] • By *Ajay Reddy* • ISBN: *013408621X* • Publisher: *Addison-Wesley Professional* • Publication date: *July 19, 2015* • Binding: *Paperback* • Estimated price: *$24.22*

Learning Object-Oriented Programming [U23] • By *Gaston C. Hillar* • ISBN: *1785289632* • Publisher: *Packt Publishing* • Publication date: *July 16, 2015* • Binding: *Paperback* • Estimated price: *$49.99*

Programming Google App Engine with Python: Build and Run Scalable Python Apps on Google's Infrastructure [U24] • By *Dan Sanderson* • ISBN: *1491900253* • Publisher: *O'Reilly Media* • Publication date: *July 11, 2015* • Binding: *Paperback* • Estimated price: *$29.62*

Reliable JavaScript: How to Code Safely in the World's Most Dangerous Language [U25] • By *Lawrence Spencer, Seth Richards* • ISBN: *1119028728* • Publisher: *Wrox* • Publication date: *July 20, 2015* • Binding: *Paperback* • Estimated price: *$27.42*

Effective Computation in Physics [U26] • By *Anthony Scopatz, Kathryn D. Huff* • ISBN: *1491901535* • Publisher: *O'Reilly Media* • Publication date: *July 5, 2015* • Binding: *Paperback* • Estimated price: *$36.74*

Programming Google App Engine with Java: Build and Run Scalable Java Applications on Google's Infrastructure [U27] • By *Dan Sanderson* • ISBN: *1491900202* • Publisher: *O'Reilly Media* • Publication date: *July 17, 2015* • Binding: *Paperback* • Estimated price: *$29.48*

Graph Databases: New Opportunities for Connected Data [U28] • By *Ian Robinson, Jim Webber, Emil Eifrem* • ISBN: *1491930896* • Publisher: *O'Reilly Media* • Publication date: *July 9, 2015* • Binding: *Paperback* • Estimated price: *$22.28*

The UX Learner's Guidebook: A Ramp and Reference for Aspiring UX Designers [U29] • By *Chad Camara, Yujia Zhao* • ISBN: *0996399801* • Publisher: *Deuxtopia, Inc.* • Publication date: *July 7, 2015* • Binding: *Paperback* • Estimated price: *$17.20*

Learn WatchKit for iOS [U30] • By *Kim Topley* • ISBN: *1484210263* • Publisher: *Apress* • Publication date: *July 2, 2015* • Binding: *Paperback* • Estimated price: *$25.59*

Python Passive Network Mapping: P2NMAP [U31] • By *Chet Hosmer* • ISBN: *0128027215* • Publisher: *Syngress* • Publication date: *July 9, 2015* • Binding: *Paperback* • Estimated price: *$51.83*

Raspberry Pi For Kids For Dummies [U32] • By *Richard Wentk* • ISBN: *1119049512* • Publisher: *For Dummies* • Publication date: *July 13, 2015* • Binding: *Paperback* • Estimated price: *$16.52*

Pragmatic Bookstuff

Here's what's up with the Pragmatic Bookshelf and its authors.

Here's what's happening at Pragmatic Bookshelf.

What's Hot

▲ 1	NEW	Mazes for Programmers
▲ 2	4	Programming Elixir
▲ 3	NEW	Rails, Angular, Postgres, and Bootstrap
▲ 4	9	Clojure Applied
▲ 5	NEW	Secure Your Node.js Web Application
▼ 6	2	Beyond Legacy Code
▲ 7	NEW	Web Development Recipes 2nd Edition
▲ 8	NEW	Your Code as a Crime Scene
▼ 9	5	Predicting the Unpredictable
▼ 10	3	Real-World Kanban
11	11	Ruby Performance Optimization

Who's Where When

And here's what the authors are up to:

2015-08-11	**GOING BEYOND THE RAILS 'DEFAULTS'** Chris Johnson (author of Web Development Recipes 2nd Edition [U1]) That Conference [U2]
2015-08-19	**Beyond "Agile" Methods** Andrew Hunt (author of Practices of an Agile Developer [U3] , Pragmatic Thinking and Learning [U4] , Learn to Program with Minecraft Plugins [U5] , and Learn to Program with Minecraft Plugins (2nd edition) [U6]) Triangle DevOps [U7]
2015-09-08	**Treat your Code as a Crime Scene** Adam Tornhill (author of Your Code as a Crime Scene [U8]) Swanseacon, Swansea, Wales, UK [U9]
2015-09-09	**Agile Creates Invaluable Testers** Johanna Rothman (author of Behind Closed Doors [U10] , Manage It! [U11] , Manage Your Project Portfolio [U12] , Hiring Geeks That Fit [U13] , Manage Your Job Search [U14] , and Predicting the Unpredictable [U15]) SQGNE [U16]
2015-09-17	**Extensions Programming Workshop** Chris Adamson CocoaConf Boston [U17]
2015-09-18	**Video Killed the Rolex Star** Chris Adamson CocoaConf Boston [U18]

2015-09-23 **Using Agile contracts in Sweden**
Mattias Skarin (author of Real-World Kanban [U19])
Upphandla IT, Göteborg

2015-09-24 **Lead organizer**
Alex Miller (author of Clojure Applied [U20])
Strange Loop, St. Louis, MO [U21]

2015-09-29 **This is a hands-on workshop where you'll learn how to leverage the capabilities of Kafka to collect, manage, and process stream data for big data projects and general purpose enterprise data integration needs alike.**
Jesse Anderson (author of The Cloud and Amazon Web Services [U22] and Processing Big Data with MapReduce [U23])
Strata NYC [U24]

2015-09-29 **Harness Your Leadership Workshop**
Johanna Rothman
Agile Cambridge [U25]

2015-09-30 **Keynote: Becoming an Agile Leader, Regardless of Your Role**
Johanna Rothman
Agile Cambridge [U26]

2015-10-05 **Treat Your Code as a Crime Scene**
Adam Tornhill
GOTO Copenhagen, Denmark [U27]

2015-10-07 **Full day Workshop: Code as a Crime Scene**
Adam Tornhill
GOTO Copenhagen, Denmark [U28]

2015-10-13 **Full day Workshop: Code as a Crime Scene**
Adam Tornhill
Software Architect 2015, London, UK [U29]

2015-10-20 **Lean Kanban conference moderator. Topic: Evolving products and organisations Fast Feedback.**
Mattias Skarin
Lean Kanban Nordic - Stop Starting 2015, Stockholm [U30]

2015-10-26 **Software Design, Team Work and other Man-Made Disasters**
Adam Tornhill
Trondheim Developer Conference, Norway [U31]

2015-11-04 **Mine Social Metrics From Source Code Repositories**
Adam Tornhill
Øredev 2015, Malmö, Sweden [U32]

2015-11-05 **Extensions Programming Workshop**
Chris Adamson
CocoaConf San Jose [U33]

2015-11-06 **Video Killed The Rolex Star**
Chris Adamson
CocoaConf San Jose [U34]

2015-11-07 **Revenge of the 80s: Cut/Copy/Paste, Undo/Redo, and More Big Hits**
Chris Adamson
CocoaConf San Jose [U35]

2015-11-16 **Coming soon!**
Mattias Skarin
Lean Kanban Central Europe 2015

What's Happening

But to really be in the know, you need to subscribe to the weekly newsletter. It'll keep you in the loop, it's a fun read, and it's free. All you need to do is create an account on pragprog.com [U36] (email address and password is all it takes) and select the checkbox to receive newsletters.

Solution to Pub Quiz

Here's the solution to this month's pub quiz:

O	Y	N	A	X	U	S	L	I
U	A	S	L	I	Y	N	O	X
X	L	I	S	O	N	A	Y	U
A	O	Y	I	U	S	X	N	L
I	U	X	N	A	L	Y	S	O
N	S	L	O	Y	X	I	U	A
Y	N	O	X	L	I	U	A	S
L	X	U	Y	S	A	O	I	N
S	I	A	U	N	O	L	X	Y

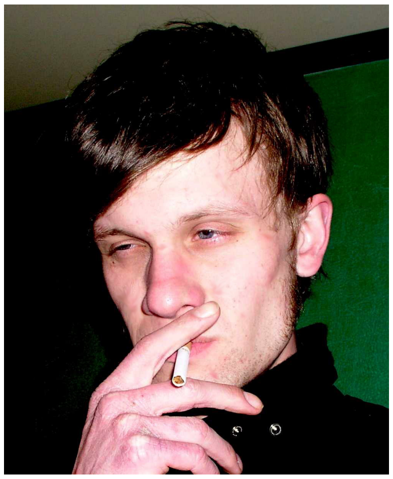
Rearranging the letters from any row, column, or smaller square reveals ANXIOUSLY. Which is how I hope you did NOT solve the puzzle.

Shady Illuminations

Ad Blockers

by John Shade

Lust, knowledge, hope, and anarchy. And ad blockers.



Jean-Louis Gassée, that's the main character in that must-read maybe-new maybe-not Harper Lee novel that nobody's reading, right? Oh, wait. Is she the large-mouthed boss in The Veep? The New Adventures of Old Christine? That Seinfeld show, whatever it was called?

Wrong, wrong, wrong. Jean-Louis Gassée, children, is the lyricist to some of Apple's cleverest tunes, the Frenchman who arrived on the shores of Cupertino with tech chops and a briefcase full of innuendo: Jacques Brel, alive and living on Infinite Loop. He was Apple's Director of European Operations and head of Macintosh development and head of Apple's advanced product development and head of worldwide marketing slash titillation and token sophisticate from 1981 to 1990 and should have given Apple more of the same as CEO but left to create a company that produced an excellent computer that didn't find its audience as well as an excellent operating system that failed to live up to its sales potential and ultimately got edged out by NeXT in the ambulance race to save Apple when it was on life support, but surely you've read the book.

Today Gassée writes astutely and eloquently about the industry in a newsletter called Monday Note [U1], but back then he wrote:

"One of the deep mysteries to me is our logo, the symbol of lust and knowledge, bitten into, all crossed with the colors of the rainbow in the wrong order. You couldn't dream a more appropriate logo: lust, knowledge, hope, and anarchy." – Jean-Louis Gassée

These days the Apple logo has lost its hope and anarchy, having no need to depend on the former and no use for the latter. Now it's all lust and knowledge.

So let's talk about ad blockers.

As Gassée explains in a recent Note, Apple is offering technology in its latest OS X and iOS releases that help independent developers create ad-blocking software for Apple's Safari browser on the Web and on mobile devices.

Ad blocking is a big deal, for two reasons.

First, ads are the business model of online publishing. Content producers, who in an earlier, simpler age were called artists and journalists and the like, have never figured out how to get people to pay for their work. Art lives off patronage. Even Shakespeare had his Earl of Southampton. And the patron always wants something for his patronage, and that something is usually you, the reader. The internet has just made things an order of magnitude worse for the content producer because the ease with which the artist's work can be copied reads like a mandate to share. The reader or listener or viewer really doesn't want to pay (a fact that Swaine doesn't seem to have figured out, by the way), so ya gotta have advertisers. If you can't sell *to* your readers, you sell your readers.

Ad blockers are an existential threat to this business model. Advertisers won't give you money if you don't deliver the eyeballs.

But then there's the other way this is a big deal. Advertisers are wrecking publishing. You've seen those websites that wriggle and twitch interminably on loading, as they keep releading to dump more ads on you. What you don't see are all the trackers loaded in the background. In another Note, Gassée's partner Frédéric Filloux reports on a test in which he visited 20 home pages and got a total of 500 trackers loaded. And you may not appreciate the full cost of the things you do see: the same Note reported a typical case in which a page loaded in two seconds with ad blocking on, and eleven with it turned off. You're thinking time because I expressed that in seconds, but if you're browsing on a cellular account, you should be translating that to a percentage of your monthly data allotment.

So you, the reader, are paying after all. You're not paying for the content you chose to read, you're paying for the ads. Read the Note. The situation is a lot worse than I've portrayed it here.

As a reader (at least that's the hat you're wearing at the moment), you probably think this situation has to change. Good news: it's probably going to do just that. This advertising model is likely to come crashing down as ad blocking spreads. Good news for readers. Bad news for content providers. Which is bad news for readers.

It's a mess. I love it.

About the Author

John Shade was born in Montreux, Switzerland, on a cloudy day in 1962. Subsequent internment in a series of obscure institutions of ostensibly higher learning did nothing to brighten his outlook. He is still waiting for Julia Louis-Dreyfus and Richard Dreyfuss to get it on, just for the fin-de-siècle frisson.

External resources referenced in this article:

[U1] http://www.mondaynote.com/