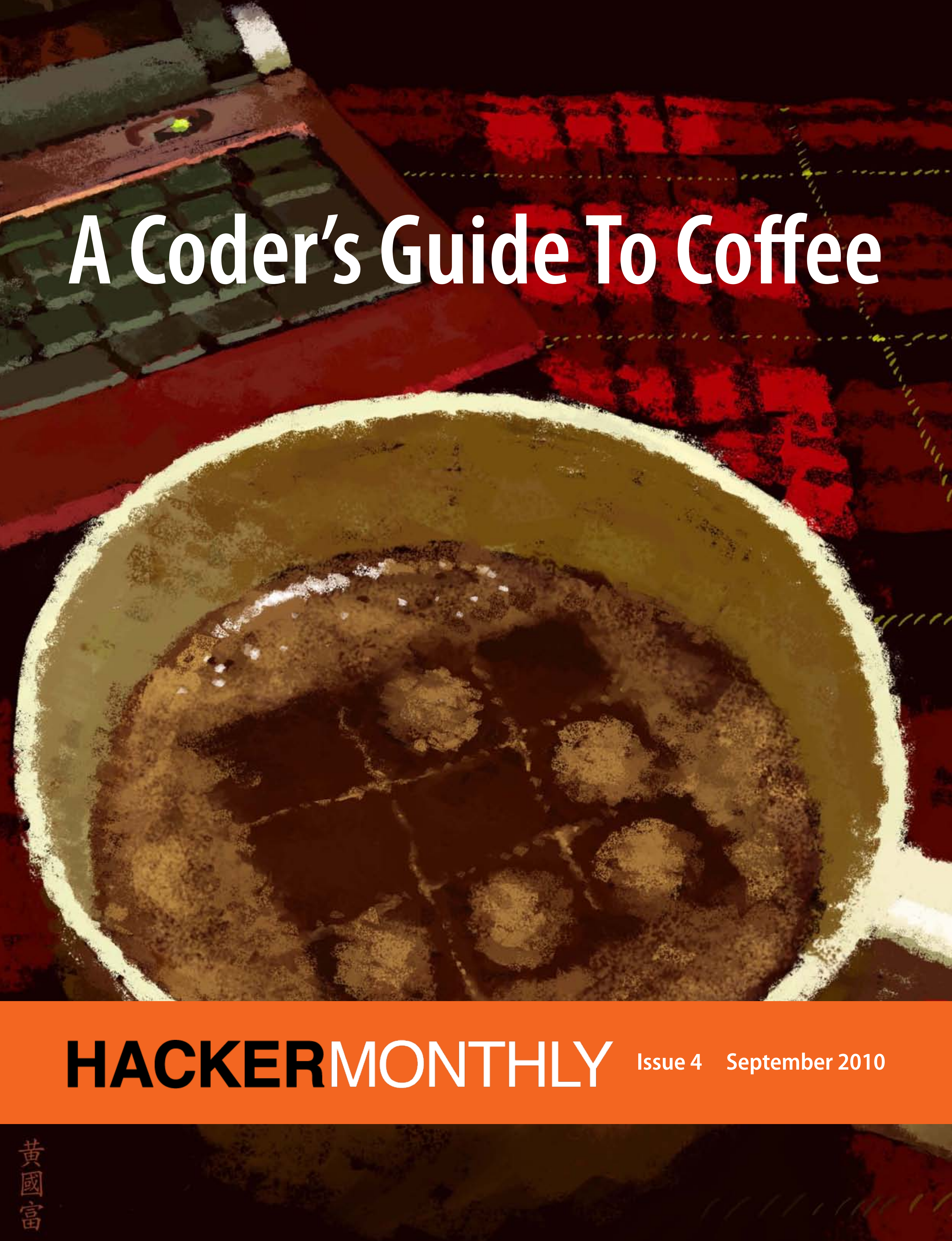
A Coder's Guide To Coffee



HACKERMONTHLY Issue 4 September 2010

Curator
Lim Cheng Soon

Contributors
Clay Johnson
Tom Moertel
Gabriel Weinberg
Sachin Agarwal
Peter Cooper
Ray Grieselhuber
Jeff Bezos
William Shields
Mark McGranaghan
Mark O'Connor
Jeff Kreeftmeijer
Mike Malone

Proofreader
Ricky de Laveaga

Illustrator
Jaime G. Wong

Printer
MagCloud

E-book Conversion
Fifobooks.com

Advertising
ads@hackermonthly.com

Contact
contact@hackermonthly.com

Published by
Netizens Media
46, Taylor Road,
11600 Penang,
Malaysia.

Curator's Note

Hacker Monthly finally has subscriptions. You can now subscribe and receive print and digital editions of Hacker Monthly for 12 months. Every print subscription includes a digital subscription.

I have also started charging for the digital format. It was not an easy decision, but I figured it is one that is better to make sooner rather than later. As I learned from the past 3 months, an advertising-centric revenue model isn't going to cut it. It seems to me that the only way to move forward is to charge for digital format.

If you are reading this, that means you have paid for Hacker Monthly. I sincerely thank you for supporting, and for helping us keep going for a very, very long time. ■

— *Lim Cheng Soon*

HACKER MONTHLY is the print magazine version of Hacker News — *news.ycombinator.com* — a social news website wildly popular among programmers and startup founders. The submission guidelines state that content can be "anything that gratifies one's intellectual curiosity."
Every month, we select from the top voted articles on Hacker News and print them in magazine format.
For more, visit *hackermonthly.com*.

Cover Illustration: Jaime G. Wong (*http://retrazos.pe/*)

Contents

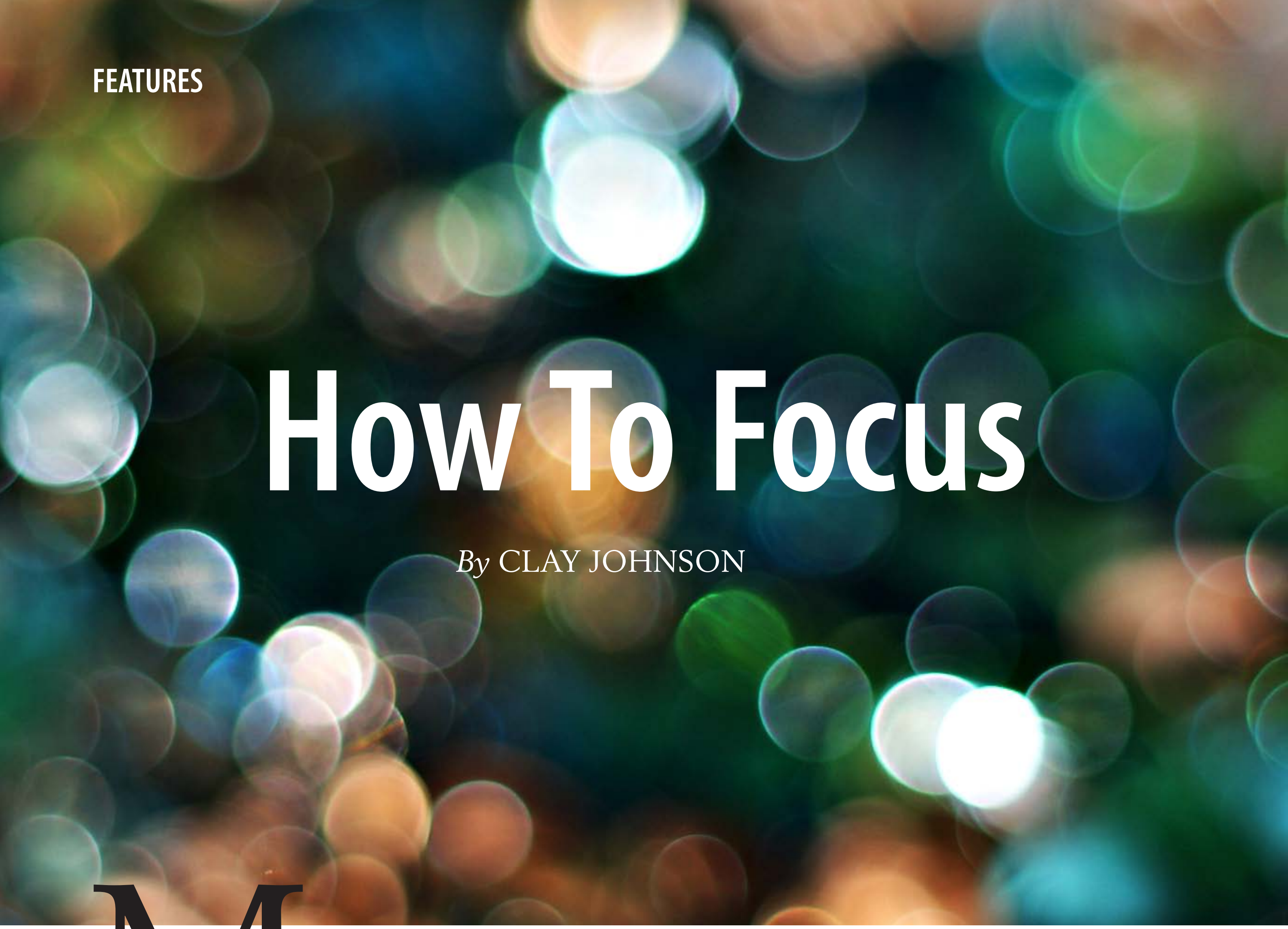
FEATURES

STARTUP

SPECIAL

PROGRAMMING

Photo: Sierra Michels Slettvet, *http://www.flickr.com/photos/sierrams/3437792169/*.



How To Focus

By CLAY JOHNSON

MOST OF THE people who click on this article from somewhere won't finish reading it. So says Nick Carr. The New York Times will remind you that you'll probably forget reading it in a few minutes. The idea has gotten so prevalent, even the Onion has started to take its jabs.

There's some truth to it. Posts like this and search trends point to what we're after. Many people want the ability to focus more and feel like they're losing the ability to focus on a particular task for long periods of time. We feel like we're losing that ability. Getting Things Done and all the other books out there tend to give you some rituals to cope with the problem — but only if you could stick to them. Most of us, just a few weeks after reading that book, sit next to filing cabinets (virtual or otherwise) and go about our merry way.

That's because we're focused on the wrong thing. To get a longer attention span — even a span long enough to read this article — don't worry about managing the information. Worry about managing your attention. Paying attention, for long periods of time, is a form of endurance athleticism. Like running a marathon, it requires practice and training to get the most out of it. It is as much Twitter's fault that you have a short attention span as it is your closet's fault it doesn't have any running shoes in it. If you want the ability to focus on things for a long period of time, you need attention fitness.

Neuroplasticity is how your brain changes its organization over time to deal with new experiences. It involves physical changes inside of the brain based on the particular tasks the brain is asked to complete. It's why the hippocampus of a seasoned taxi driver in London is larger than average, and how a meditating monk grows grey matter. Your brain isn't a mythological deity but a physical part of your body that needs to be taken care of just like the rest of your body. And your body responds to two things really well — diet and exercise. Let's presume your brain, being a part of the body, also does.

Things like Inbox Zero or cutting down on meetings may be handy tricks, but they don't take neuroplasticity into account. The bet there is that you have a finite amount of attention to spend, and that attention range isn't changeable. That stuff is handy for making the best use of your limited attention span, but it's not going to improve your attention span. It's not going to stop your brain from being easily distracted or unfocused if you've already trained it to be that way.

Photo: Pranav Sankar Yaddanapudi, *http://www.flickr.com/photos/neychurluvr/3611966967/*.

> “Modeled after how I trained to run my first marathon using Jeff Galloway’s technique, I practice attention interval training.”

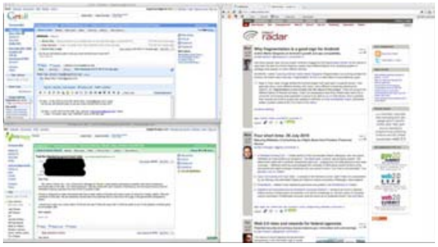
So how do you train to focus? I’ve been using interval training with great success. Modeled after how I trained to run my first marathon using Jeff Galloway’s technique, I practice attention interval training. I got this timer installed on my computer. It’s an excellent interval timer based on a technique called the Pomodoro technique — but I’m primarily using it based on its ability to make sound, set good intervals, and support logging. I started small: 10 minutes of work with two minute breaks. My strategy has been to keep it so when the timer goes off that tells me it’s time to take a break, I feel like I can keep going. I’m up to 35 minutes now with 2 minute breaks. Interestingly enough — this is about as far as I’ll get probably while still being able to keep Instant Messaging on. I’ve found that about 35 minutes is the max response time for IM to be useful.

The timer isn’t the key part though, that’s just a component of a system like a good watch is a part of running a marathon. Here’s how I set that up:

Ditched the Second Monitor

I’ve been using a second monitor for nearly ten years, thinking that vast amounts of space were key to productivity. The second monitor myth has been around for quite some time. Yet the only actual scientific study I could find linking multiple monitors to productivity was done in 2003 by a monitor manufacturer, a video card manufacturer, and the University of Utah. It’s actually kind of a marketing document, not a study. I’ve opted for one, large monitor. Two monitors just allows me to put distractions on one monitor, and actual work on another.

Set up Spaces in OS X



Spaces is virtual desktop software on OS X. I never thought it was useful before ditching the second monitor, but now — instead of having always-on distraction in one monitor on my desk, I can put my email, twitter, and surfing browser in one “Space” on OS X and keep it there. When I start my pomodoro timer, I hop into a “space” that has only the tools I need to do whatever task I am up for on the screen. In this case, I need limited web browsing and a text editor to write this blog post. Note the addition of “about:blank” in my bookmark bar at the top of the browser. While I’m writing and don’t need to use the browser, I tend to blank the screen out so I don’t get too distracted by the browser.

My third space simply has Remember the Milk running in full screen.

Turned the mouse off during work-time

During the time that I’m working (unless I’m editing) — my 35 minute work intervals — I turn my mouse off. I’ve found that I can focus much more on the task at hand if I don’t touch or use that mouse. For me, my mouse is a gateway towards passive browsing and web surfing. If I don’t have access to it, I can’t begin the chain reaction of getting sucked into the web. For me, it’d be like running a marathon on a road with 26.2 miles of chicken-wing stores. I might make it a few miles, sure, but around mile 20, I’m going to succumb to temptation. I’ve found that Divvy helps me manage windows without the mouse, and that Vimium helps me use the web for research without the mouse.

Created a proactive routine

Part of my 2 minute break-time is used to set-up whatever tools I need to accomplish my next task. I use that time to figure out exactly what I need for my next task, close-down all the things I don’t need for that task, and set windows up appropriately. There’s rarely a time when I need more than two windows open. The set-up generally involves closing all tabs in the browser, and starting the browser fresh with an about:blank page. The key here is, I don’t just hop into doing work. I spend a minute or two setting up an ideal environment for me to be able to complete whatever my next task is. When I leave my computer for the day, there are no windows open. I start with a blank slate to come back to. No need getting bogged down in yesterday’s set-up.

> “Do slightly less than you think you’re capable of.”

About those tabs

None of my web browsers — surfing or otherwise, are allowed to have more than 5 tabs per browser window at any time. I do this via the No More Tabs Chrome extension. This extension is pretty brutal: if you create a new tab and you’re over your tab limit (defaults to 5) it’ll close your oldest one. I’ve been running this extension for over a month, and not once have I had a serious problem. It’s forced me to pay attention to a particular web page and finish working with it if I’m going to move on to something else.

The Environment Around Me

While I work primarily from home, I’m still prone to distractions from my environment. To conquer that, I have a pair of noise-reducing headphones, and I listen primarily to lyric-free music. Just a bit of noise to keep me focused. I sit at my desk, but I suspect that I’ll be converting to a standing desk soon because I don’t want to die early. I also tend to keep some snacks (nuts) and beverages around my desk so that food and water don’t lower my focus threshold. Though there’s one big anomaly here: I’m not working in an office with that many people in it. I don’t have a lot of meetings to take. I’m not managing anyone right now. For that though, I suggest consolidating all meetings into the afternoon and make them back to back. That way, you’re getting them out of the way and you have solid, long blocks of time to focus on getting things done.

Synopsis

Like all exercise, different kinds of workouts work differently for different people. For me, interval training works wonders — this blog post, for instance, has taken me 70 minutes to research and write — ordinarily a blog post like this before I had this set-up would take me nearly a full day’s worth of work. More importantly though, I’m able to do things like read long articles or even academic papers — things I never used to “have time for” which really meant “had attention for.”

If you think you’re having focus problems — if the concept of provigil appeals to you, or you’ve thought “oh if I could only get my hands on some ritalin,” think about setting up an attention fitness regimen for yourself instead. My general advice:

1. Do slightly less than you think you’re capable of.
2. Increase your capacity while staying under that bar (#1).
3. You’re not going to run the attention fitness equivalent of a marathon today. Start slow.

Your brain, like your body, is only a result of what you train it to do. Attention fitness, like any other kind of fitness, takes time even to get into a routine. But once you make it a habit, it starts to pay off. ■

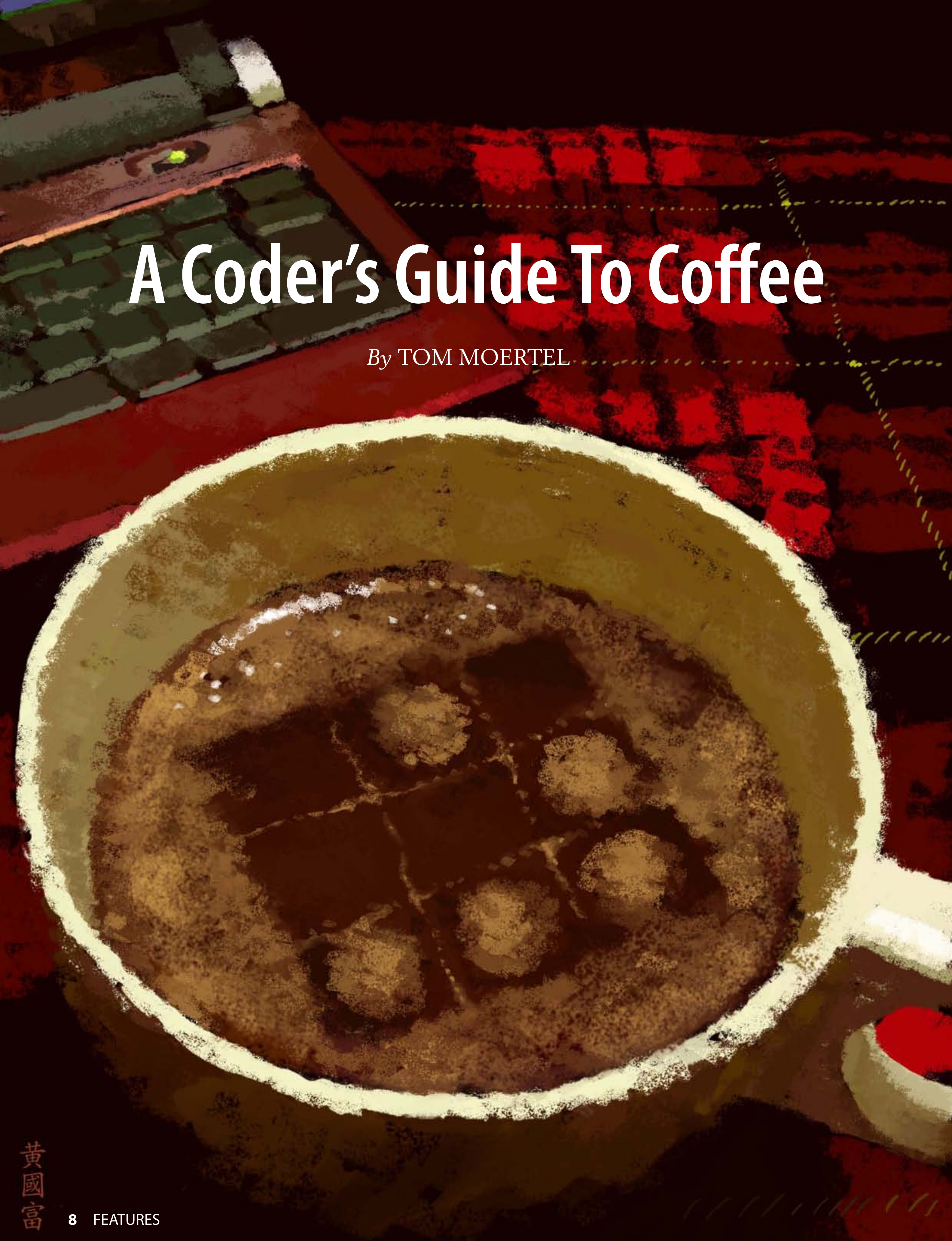
Clay Johnson is an open government advocate. He is the former director of Sunlight Labs and under his leadership, Sunlight Labs became a community of 2000 developers and designers working to open their government. Prior to Sunlight, Clay was one of the four founders of Blue State Digital, the organization that provided the technology and online strategy behind Barack Obama’s presidential campaign. He blogs at *infovegan.com*.

Reprinted with permission of the original author. First appeared in *http://hn.my/focus/*.

Duck Duck Go

A Coder’s Guide To Coffee

By TOM MOERTEL



As most software and creative professionals know, coffee is an important technology for boosting mental acuity and maintaining peak on-the-job performance. But did you also know that coffee can be a damn tasty beverage? It's true. All you need is the appropriate amount of disrespect for the mainstream coffee industry and a desire to enjoy a better beverage. So read on, and learn the secrets to great coffee.

First things first. Mainstream coffee sucks, and specialty coffee mostly sucks. Mainstream coffee is primarily stale, low-quality, high-yield beans, many times cheap robustas, foisted on a largely unknowing public in supermarkets nationwide. Specialty coffee isn't so much coffee as it is flavorings, sweeteners, and milk; what coffee is sold is often neither "special" nor properly prepared – it's usually over-roasted to serve as a background for sweet flavorings. A few specialty coffee purveyors, however, do sell good coffee, and I'll show you how to find them, but most are happy to sell you stale beans whose dead taste is hidden behind raspberry and caramel syrups. Buyer beware.

Nevertheless, good coffee is good – great even – all by itself. It's also dirt cheap and easy to make. Therefore, don't settle for a cup of crappy coffee: make a cup of the good stuff for yourself.

The coffee quick course

If you follow these three guidelines and do nothing more, you will enjoy coffee better than you can find in most specialty coffee shops:

1. Buy only whole-bean coffee roasted within the last few days.
2. Grind it fresh, just before brewing.
3. Brew it in a French press or a pour-over filter using fresh water, off the boil.

The first two guidelines strike at the nemesis of good coffee – staleness. Stale coffee is dead coffee. There is no way to get a good cup from it.

Sadly, most of the coffee you buy in stores is stale before you get it home. While green (un-roasted) coffee beans can stay fresh for 2 years, roasted coffee goes stale in under 2 weeks, and ground coffee goes stale in a few short hours because of the immense surface area that grinding exposes to the air. Special "freshness preserving" packaging doesn't help much either; it's mainly a marketing gimmick.

The only reliable way to get fresh coffee is to know when it was roasted. Therefore, when you buy coffee, buy it from a purveyor who can tell you when it was roasted. If a coffee purveyor can't or won't tell you when their coffees were roasted, find another purveyor. And when you buy your coffee, buy it whole bean. Store it away from heat and light (but not in the refrigerator). Use it before it goes stale. If it goes stale, throw it away and get fresh beans.

Also, get a grinder. An inexpensive ($15) blade grinder ("whirly-bird") is sufficient for making drip coffee and lets you grind just before brewing, which is the key to avoiding staleness. At this price, there is no reason to suffer stale, pre-ground coffee. If you want to buy a better grinder, that's fine, but don't think you have to spend a lot of money to enjoy fresh coffee.

The third guideline addresses another common flavor-denial attack: Low-temperature brewing. Most drip coffee makers brew at a temperature too low for proper flavor extraction. The most frequent explanation that I've heard for this sad yet pervasive flaw is that "really hot" coffee is a lawsuit waiting to happen, and thus manufacturers have lowered brewing temperatures accordingly. Whatever the reason, the effect is a cup of lifeless coffee.

So what is the right temperature? Off the boil works well. Put a kettle of freshly drawn, cold water on the stove. When it boils, take it off the heat, wait a minute or so, and slowly pour it over your freshly ground coffee. If you're an experimenter, a $10 instant-read thermometer is all you need to "dial in" the optimal temperatures for your coffees and your taste-buds.

Since you'll be using a "pour over" technique, you'll need a pour-over brewing apparatus – either a French press or a $5 pour-over filter holder, found in most supermarkets. Use the French press if you enjoy the stronger flavors of unfiltered coffee. Use the filter holder if you prefer the convenience of a filter, which makes clean-up easy. Both are small enough to take to work, and the filter holders are cheap enough to leave there.

And that's how you make great coffee. If you think that's too much effort, at least you can use your new knowledge to find coffee shops that use fresh beans, grind them just before brewing, and brew them properly (most commercial brewers do use proper temperatures, thank goodness).

Oh, there's more

If you follow the advice above, you will drink great coffee for the rest of your life. For some people, that's enough. For other folks (like me), that's just the beginning. It's the first step toward a fun, inexpensive, and gastronomically rewarding hobby. Even if you don't want to make coffee into your hobby, you do have the opportunity – right now – to give up bad coffee and start drinking the good stuff. Why not seize the day?

Illustration: Jaime G. Wong (*http://retrazos.pe/*)

01

02

Home roasting

Roasting your own coffee is simple and provides three major benefits. First, you can buy your coffee green and store it for over a year. Second, you can roast your coffee as you need it, so you'll always have fresh beans. Third, you can experiment with a wide variety of beans, blends, and roasts to enjoy coffee that you could never find in a store.

A further benefit is that green coffee is less expensive than roasted coffee. By home roasting you'll not only have better coffee and more control but also more money in your pocket.

To roast your own coffee you will need two things: green beans and a roaster. The beans can be purchased online at places like Sweet Maria's (where I get most of my beans) and locally from the better coffee shops in your neighborhood. A roaster can be had for as little as $5 – buy an old hot-air popcorn popper at a garage sale. That's what many folks on alt.coffee use for their roasting. If you prefer a less adventurous solution, there are many home-use roasting machines now on the market in the $100–$300 price range. I use a $150 Hearthware Precision roaster, and it works well. Just drop in a scoop of beans, dial in the desired roast, and press a button.

Yes, it's that easy. And, yes, the results are better than most pre-roasted coffees you can buy. Nothing smells as good as freshly roasted beans. Nothing tastes as good when brewed. Once you try home roasting for yourself, you will understand.

Espresso

If you want to experience the concentrated essence of coffee, you must drink espresso. Good espresso. Unfortunately, practically none of the specialty coffee shops and chains in the United States knows how to prepare espresso properly. If you want good espresso, you'll have to make it yourself (or take a trip to Italy).

Unlike the advice I provided earlier, which is simple and just plain works, making good espresso is difficult. Finding the right combination of beans, grind, packing, pressure, temperature, and exposure takes practice. It took me months of gradual refinement to learn how to make a truly good cup. After years, I'm still seeking the perfect cup.

Since the perfect cup of espresso is a never-ending quest, I can only point you in the right direction. The rest is up to you. Here is what I can tell you:

- Plan on spending > $250 USD on a good pump machine. "Steam toys" aren't capable of good espresso. Do your homework: Read what people who own the machines say in the consumer reviews of brewing equipment on on CoffeeGeek.com.
- Plan on spending that amount again on a good grinder. Many people buy an expensive espresso machine but skimp on the grinder. Big mistake. Since grind

Photo: GenBug, *http://flickr.com/photos/genbug/3153754889/*. Sierra Michels Slettvet, *http://flickr.com/photos/sierrams/3437792169/*. Alex, *http://flickr.com/photos/eflon/4340500871/*.
Licensed under Creative Commons Attribution 2.0 Generic licence. Full terms available at *http://creativecommons.org/licenses/by/2.0/deed.en*.

03

is probably the single most important variable under your control, a grinder must be highly adjustable and produce a consistent grind, and that means high-quality burrs set in a rigid enclosure. These features don't come cheap. When shopping for a grinder, again, check out the consumer reviews on CoffeeGeek before buying.

- Read the alt.coffee wisdom on espresso, ristretto, crema, and tampers. It's also a good idea in general to hang out on alt.coffee. I've learned most of what I know about coffee and espresso there.

If you want to dig a little deeper, you can read my semi-rant about espresso on Slashdot (http://hn.my/coffeerant/).

Brew well and drink well, my friends

Although coffee is commonly considered a utility beverage, it is an amazing drink when well prepared. Given its ubiquity in software and creative circles, it's likely that you will be drinking a lot of it. So why not prepare it as it was meant to be? Why not enjoy a cup of truly good coffee? If you buy fresh, high-quality beans, grind them on the spot, and brew with hot water, you can't go wrong. And if you decide to try home roasting or espresso, you will enter a whole new world of flavor and nuance. The rewards are worth the effort.

Whatever else you may do, please don't let the mainstream coffee industries convince you that bad coffee is all there is. Good coffee is out there. Insist on the good stuff. ■

Tom Moertel designs and codes the core messaging systems at Smash Technologies (www.smashcode.com), a mobile-messaging startup based in Pittsburgh. To make things go, he uses Erlang, Haskell, Python, and a whole lot of home-roasted coffee. You can read his blog at *blog.moertel.com*.

1. Coffee Beans
2. French Presses
3. La Pavoni Espresso Machine

Originally posted to *Kuro5hin.org* on Thu Apr 25th, 2002 at 02:29:34 PM EST. Reprinted here with minor corrections as it appears on *http://hn.my/coffee/*.

Wannabe Entrepreneur Symptoms And Cures

By GABRIEL WEINBERG

I WAS ONCE A wannabe entrepreneur. Fresh out of college and a summer internship at a VC firm, I thought I knew what I was doing. Though this was 2000, and all startup & VC blogs we've grown to love didn't exist yet, I did have mentors available. I should have leaned on them a lot more, but I didn't, or at least not in the right ways.

But all the ways I've failed, and there are certainly many, is not the point. I just want to let you know that I've been there, and that I hope the rest of this post doesn't come off as annoyingly condescending.

Since 2000, I've been doing and thinking about startups constantly. Even though I'm an introvert, I end up meeting or otherwise crossing paths with a lot of entrepreneurs. Unfortunately, I'd classify a lot of them as wannabes.

What follows are some symptoms I've seen over and over that usually (though not always) indicate a wannabe entrepreneur. If any of these describe you (or someone you know), I'd take it as a sign to step back and think hard about what you're doing (or have that conversation with your friend).

There are cures. Usually it means what you (or they) are working on now will fail. But perhaps it is salvageable with a few tweaks or a change in direction. And if you/they are really in it for the long term (as real entrepreneurs are), then there will be other startups.

Symptom: a year has gone by and you have nothing to show for it.
Cure: get stuff done. That's what real startup founders do. Customers don't care about excuses.

Symptom: you haven't really talked to any real customers/users.
Cure: read Steve Blank's book. Get out of the building. "No plan survives first contact with customers." A related (non-wannabe but first-timer problem) is confusing the user with the customer. I did this on my first startup, and it was one of my primary problems.

Symptom: you're going around calling yourself a CEO.
Cure: you're a founder. You're not powerful. No one cares about what you're doing...yet.

Symptom: you aren't knowledgeable about startups, especially your own space.
Cure: read stuff & regularly talk with the smartest startup people you know. At the very least, you should know the whole history of your space--failures, acquisitions, IPOs, reasons for such, etc.

Symptom: you just need 10-25K in investment.
Cure: get your own 10-25K. Do consulting. Maybe convince friends and family. If you can't raise that much from yourself and your existing circle, you aren't going to be able to raise more from strangers. I did consulting for a few years, max 4 hours a day, so I could focus the rest of time on my startups.

Symptom: you have spent months researching the right architecture to build your site.
Cure: build it already. You seem like someone more interested in technology than startups.

Symptom: you don't understand your startup's assumptions.
Cure: make a spreadsheet and try to predict the key metrics of your business. Yes, the financial projections that come out of the spreadsheet are probably worthless (or grossly inaccurate), but not their underlying assumptions. Those are the things you need to prove and the first step is knowing what they are. As a side note, this exercise will help you understand how much money you need to raise, if any.

Symptom: you've written more than a 5 page business plan (intended for others).
Cure: spend that time talking to real customers or building your product. If you think it will help you understand your business, build a spreadsheet with assumptions instead. If you think investors will read it, know that they won't. Note: I have no problem with people analyzing their businesses internally through brief writing; I do that too.

Symptom: you now just need a programmer to code up your site.
Cure: either convince a real tech co-founder to join you, or learn how to code yourself. It's not that hard, and if you think of startups as a career, it's a great skill to have even if you just manage tech people. You don't have to major in CS in college to be a programmer, e.g. I was a Physics major. ■

Gabriel Weinberg is the founder of Duck Duck Go, a search engine. He is also an active angel investor, based out of Valley Forge, PA. More info at his homepage: *http://ye.gg/* .

Reprinted with permission of the original author. First appeared in *http://hn.my/wannabe/*.

If You Can’t Buy Your Investor a Beer, Don’t Take Their Money

By SACHIN AGARWAL

We just had our second official board meeting. Posterous has been around for over two years, but I still count this as number two. Because in the past, a board meeting just meant Garry and I were at a bar talking about the future of our company.

But there were new faces at this meeting. Satish Dharmaraj from Redpoint Ventures, Gus Tai from Trinity Ventures, and our lawyer, John Bautista from Orrick. There was no beer. And I was presenting the state of Posterous through a Powerpoint presentation. I had barely touched Powerpoint before starting Posterous.

The guys around this table have a lot of power over our company. They are on the board. They have voting rights. They can fire me.

So how do we know we picked the right people for the job? Some of the VCs we were pitching to, we met only three or four times. Is that enough to really get to know someone, to give them power over your company and future?

Do they know me? Do they know what my goals are, what kind of company I’m trying to build, what gets me excited?

There are plenty of posts online about valuations, term sheets, and how to negotiate. I’m not going to get into that stuff here. This post is about the personal side of finding investors. These are tips to make sure the people you let invest in your company are a good fit.

You need to trust your investor, and you only have a short amount of time to find out if you do. Here are some things Posterous did to get to know our investors before letting them invest.

Get to know the VCs early. If you need money, you are too late.

If you aren’t raising money, you have the luxury of time. Use it to meet and get to know as many VCs as possible.

1. Go to startup events and introduce yourself to every VC in the room.
Don’t just do a five second hello. Tell them who you are and what you’re working on. Even if you aren’t looking for money, they will appreciate meeting you.

2. Read VC blogs.
What VCs do you look up to? Find out what every VC writes about, what his beliefs are, and what he’s invested in.

3. Beware of associates.
We had some bad experiences. If an associate sets up a meeting with you, make sure a partner will also be there.

4. Get introductions to VCs from your angel investors and other startup friends.
These go a long way. If you have multiple connections to a VC, have them all plug you. If a VC hears about your company from 5 of his friends, he will meet with you.

5. Don’t be shy.
Be proud of what you’re building. Your competition will be. Highlight your strengths, be confident.

6. A couple months before you’re going to raise, schedule a coffee meeting with all the VCs.
No pitching, no deck. We did this and it was a great way to meet VC partners in a more casual setting. If they like you, they will even help you with your pitch.

7. Continue building a relationship with VCs you meet.
Send them updates about your company, news in TechCrunch, and updated stats.

8. Get them to use your product.
If they haven’t used it by the time you’re pitching them, you’re wasting your time.

Let the pitching begin

You are not cattle. Make the VCs respect you and your time.

1. Cram all your meetings in the shortest amount of time possible.

You want to get this over with quickly. (You also want your term sheets to come at the same time).

2. Refine your pitch everyday.

Figure out what works and what doesn't, then change it.

3. Put your least desirable VCs up front.

You will learn a lot as you go. You will figure out which questions are good, and which are signs of interest.

4. Don't read too many posts about what a VC pitch deck should look like.

You know what it should contain? Whatever you want it to. Because it should be personal. It should convey what you think it important. Otherwise you might try to squeeze your pitch into a mold that isn't right for you.

5. Your pitch should be natural.

By the middle of our pitching calendar, I could give our pitch by heart and it was 90% the same as the last time I gave it. That's not because it was memorized. It's because it was natural and automatic.

6. Before starting your pitch, make everyone in the room introduce themselves.

Sometimes they don't and it's very odd. They should be selling to you as well.

7. Get through your pitch and divert as many questions as you can.

You should run the show.

8. Have one person speak, whoever is the most confident.

It will flow better this way and you won't be repeating yourself as much.

9. When asked a question, have one person ready to answer it.

Don't look at each other, don't hesitate.

10. Ask the VCs questions.

Have these ready beforehand. Ask them about the firm structure, their funds, and other investments.

11. Evaluate the VC's questions.

Are they asking you smart things? You can pretty accurately figure out if the VC "gets" what you're building and is excited about it based on their questions. If they don't get it, don't waste your time.

12. If they don't get it, walk out.

Say, "no thanks." I said "no" to a couple VCs when I thought there was no fit. I don't want to waste their time, and they shouldn't waste mine. If you ask me about barriers to entry, you don't understand the internet.

13. Follow up with a thank you email, and additional questions.

This is your chance to ask anything.

So you got a term sheet. But do you want their money?

If you're fortunate enough to get multiple term sheets, here's when you decide which VC you want on your board. Terms are important, but your fit with the partner will mean much more at the end of the day than a higher valuation.

1. Hang out with the partner over beer.

The more beer you have, the better. If you are going to work with this person for years to come, you have to be comfortable around them.

2. Check up on references.

Talk to other companies they have invested in. Ask friends who might know people at those companies. Try to figure out which references are honest and which are just siding with the VC by default.

3. Talk to CEOs that have been fired by this VC.

You'll get a good story at least.

4. If the VCs are prepping their references for your call, be afraid.

Our best offer to check references came from Gus when he said, "Feel free to contact any person I have ever worked with through my entire career."

Raising series A financing is one of the most stressful, unique, and exciting things I've done at Posterous. I had the time of my life. It was a two month long roller coaster of meetings, negotiations, dinners, and eventually, money.

We couldn't be happier with the ways things turned out. We love having Satish and Gus invest their money, time, and expertise into Posterous. In fact, the reason why we have two VCs is because we wanted them both!

We have a long relationship with Satish, and I trust him like he's part of my family. We often call him Uncle Satish. The first time I met Gus, months before we were ready to raise a VC round, I was instantly impressed by him. Kate met Gus at a Christmas party, and after just a few minutes, commented about how great of a guy he is. These things matter.

So what are our board meetings like? We rush through the legal and finance business as quick as we can, and then we talk product. We all love talking about Posterous and what we should build next. I love it. ■

Sachin Agarwal is the cofounder and CEO of *Posterous.com*. Prior to starting Posterous, Sachin spent 6 years working on Final Cut Pro at Apple, Inc. Sachin graduated from Stanford University in 2002 with a degree in Computer Science. You can follow Sachin on twitter at *http://twitter.com/a4agarwal*.

Reprinted with permission of the original author. First appeared in *http://hn.my/investors/*.

THREE YEARS AGO I SOLD MY STARTUP BECAUSE I WAS AN IDIOT

By PETER COOPER

THREE YEARS AGO today, I sold the assets forming Feed Digest to Informer Technologies, Inc. It has since been rebranded to Feed Informer but is still operating mostly as it was.

I made a nice amount of money from it and I don't regret the sale, but, ultimately, I was dumb for letting Feed Digest to get to a position where it was better to sell than not. I need to go into some background for you to see why.

The genesis of Feed Digest

Feed Digest was a pioneer in "serious" RSS (and Atom!) feed manipulation and syndication services. It wasn't the first, but it was the first to seriously try to capitalize on the idea rather than offer these services as an "extra" on something else.

Users included NASA, the Smithsonian Institute, MIT, and, most importantly, the Denver Post, one of America's top 10 daily newspapers, who used Feed Digest widgets all over their site. About 250 million "digests" were being served per month by mid 2007 and from the outside, it looked like a promising business.

Feed Digest spawned from a side project called RSS Digest whose main goal was to take my Delicious links and automatically put them onto my blog. After hundreds of others though it was a great idea, I started to take donations (several thousand dollars in a few months – not bad for 2005!) and let people use the tool too. From this, enough promise was shown that I decided to "go pro" and a month before launch, Kelly Smith structured a deal for Curious Office to make an angel investment in my idea (for this gamble on his part, I will always be grateful).

A popular service, headed by a business dunce

The nascent business had a problem though – me. Though I'd been working for myself for 7 years before launching Feed Digest, it would be an understatement to say I was naïve about the concerns of "running a business." Feed Digest's pricing was ridiculously low (even the biggest "enterprise" customer was only paying $200 a month – most users were $25-50 a YEAR) and while it was in profit from the third week and had a flood of customers throughout, it was only making several thousand dollars a month after 2 years. Enough to keep it going as a "job" for me, but not a serious business that warranted further investment.

As with most webapps, a lot of hope was put into developing an all-magical "version 2" and I made good progress with it. In 2006 I got an e-mail from Michael

Arrington who wanted to write about Feed Digest for TechCrunch (yes, one of the biggest and most influential tech blogs in the world). Being a closet perfectionist, I asked if I could hold until the magical version 2 was released. A crazy mistake. Michael has been kind enough to follow me on Twitter and occasionally throw a comment my way in the years since, I hope I can make up for my stupidity by giving him a good story one day ;-)

Service was always a massive deal for me. What I lacked in business acumen, I made up for by delivering the best service I could. Every e-mail got a response within hours, every criticism either resolved or accepted gracefully. I forget the exact numbers but in well over 1000 transactions there were certainly fewer than 5 refunds ever requested (and all were given). Users genuinely loved the service.

The sale

Service is only one leg of the stool in any business, and without a serious idea of how to grow the business it was destined to stay small fry unless someone else took notice. In mid 2007, I got a terse e-mail from a Russian guy asking if I wanted to sell Feed Digest and, if I did, his company would be interested in buying it. Negotiations were quick and my angel investors – who were supportive, but a little forlorn over my inability to drive it as a "real" business – gave the thumbs up, since the number mentioned ensured they'd get a healthy return on their investment. The deal was sealed by the end of July 2007 (no big story here, both sides did due diligence on each other and a good escrow was used) and the money hit my bank account in mid August.

It wasn't a lot of money. Well, it was for me. It wasn't a million dollar acquisition, but certainly more than most of the population will earn in a couple of years. A low six figure sale. For something merely making "a living" and keeping me up all hours of the night, I think it was a good deal. It turned into some healthy savings, a wedding, a car, and a chunk of the house we live in now, so I can't complain.

The contract I signed with the buyer meant that I was contractually obliged to defend them against any third party claims over patents, copyrights, and the like, for three years, ending July 30, 2010. Today, then, is truly the day all ties with the service are severed. It's the final period on the last page of the book.

What I've learned since

A benefit of having some savings is that I've been free to do mostly whatever I like since then. I've tried a lot of crazy ideas, got into pro blogging (with Ruby Inside), read a ton of books, worked on open source, and devoted a lot of time to learning and experimenting with business. I've got to know a lot of people in business and get a feel for how things really work, if just at a high level.

One finding is that many people running startups are no smarter than I was when I started Feed Digest. The difference, though, between me and the successful founders are that they are good at delegating responsibilities for areas they're not strong at, whereas I took a strong DIY approach, building not only the technology, but keeping books, design, marketing, and so forth. Not only that, but I wasn't brave enough to charge a sensible rate for the product and didn't understand that I should define my market rather than sell $11.99 annual accounts to people adding widgets to their personal homepages.

I've not become a Donald Trump, but I can now see what I was doing wrong and how I could have turned Feed Digest into a "serious" business (where "serious", at my kindergarten level, means enough revenue and growth to scale to multiple employees and make serious inroads to enterprise deployments of our services).

Not quite the end

Despite my Feed Digest story ending, it has caused me to reflect. I've written about the business side of things here, but the technical opportunities have changed significantly since early 2005 too. Building a Feed Digest style service now is incredibly easy.

Making something seriously powerful and sellable at an enterprise level is no longer prohibitively expensive, just dependent on having the right know how. Which… I have. I've been building feed processing and manipulation systems in the background for other projects (such as coder.io) in the interim. This has only just made me think: am I crazy for not getting back into the feed processing and manipulation industry again?

So, I'm investigating it – slowly. I know what I'm doing tech side. I have enough contacts now to get things rolling. And I have a lot more grace and life experience to actually ask for help and be brave in making business decisions than the 23 year old me ever did. The industry is still small but, importantly, still growing (Superfeedr is a notable new player) and still considered important by the right people behind the scenes.

Should I get back to what I know and build a "serious" high quality product or service on my existing knowledge base? Honestly, I don't know, but it's going to be interesting trying to find out. After three years, it no longer feels like an unethical thing to consider.

Added: I want to take this opportunity to thank Marshall Kirkpatrick – now of ReadWriteWeb – who championed Feed Digest quite a lot in its early days. He is easily the most RSS-obsessed person I know this side of Dave Winer. ■

Peter Cooper is a UK-based entrepreneur, author, and founder of *coder.io*. His personal homepage and blog is at *peterc.org* and you can follow him on Twitter at @peterc.

Reprinted with permission of the original author. First appeared in *http://hn.my/soldstartup/*.

The Royal We: Single

By RAY GRIESELHUBER

IT HAS BECOME a common wisdom of sorts in the startup world that if you are running your business without a co-founder or partner, the odds are stacked against you. I personally don't believe we have enough data to say whether the odds are any worse as a single founder, but I do know it comes with its own unique challenges. More and more, though, I meet people running their businesses, quite successfully, as solo founders. It is at least partly related to the shifts we are seeing in startups as a whole. I've learned quite a bit over the last two years as a solo founder myself, so I thought I'd share my observations and some techniques that I've found useful for making it work.

But first, let me provide some background. If you happen to follow the world of startups, especially web startups, you'll know that venture capital is undergoing a change and more startups are being run on less initial investment and that the size of each investment is getting smaller. At first glance, this may not seem like a good thing but it is for the following reasons.

First, it creates an environment that selects for do-ers and makers — people who have the ability to create entirely new businesses literally with their own hands. In many sectors, the startup world consisted of all-star executive teams and millions of dollars in venture capital. We are now discovering that this is a dangerous model for many reasons (which I won't discuss here.) But now, as companies raise less money early on, this forces them to spend more time discovering what works as a business model before they build out their product.

Second, the earlier a startup raises money, the more risk they are asking investors to shoulder. As a reward for bearing this risk, investors generally get more control of the company. By being able to build an initial product and discover a working business model with less investment, this shifts the balance of power (and the risk, of course) back to startup founders.

One of the key enablers for this shift is the dropping cost of building businesses in the first place. Paul Graham is well-known for articulating this as being due to open source software and the cloud. I also think it's fair to say that the tumultuous economy we've experienced over the last 10 years has contributed to the creation of a generation of frugal, scrappy entrepreneurs.

If we take everything above into consideration and look honestly at the difference between single founder startups and startups run by cofounders, we can see that these forces are relevant to anyone building a new company, regardless of team size. In fact, one could probably argue that if startups are going to keep getting cheaper to build, then it only makes sense that the minimum size of a team necessary to build a new startup should continue to approach zero. One is closer to zero than two or three.

Founder Startups

Kidding aside, if we can agree that these factors apply for any company, then there must be a different set of reasons why people think the odds are better for startups with co-founders.

The three that I think are most relevant are 1) emotional fortitude, 2) having more hands to do the work and 3) a richer source of new ideas. Working as a solo founder means that you need to get creative about how to make these three factors work for you, despite your status. It may be more difficult to do so, but if you can learn to do this on your own, then you will probably emerge stronger and better than many other startups, even if they are bigger than you.

For #1, there is no getting around the fact that running a startup is hard. It will test everything you know and believe about yourself. You will feel stupid, under-appreciated, underpaid and both emotionally and physically drained. And that's just what happens when things are going well. (If you're not feeling those things at least some of the time, you're probably not to the stage of validating your business model.) So, perhaps the best reason for having a co-founder is for moral support, someone to help you get through those dark nights of the soul.

As a single founder, I can't argue that having the right co-founder would not make this much easier. But there are ways of making it work, even if you are alone. The key is to not be alone in other, even more important areas of your life. In my case, my wife helps me get through. In many ways, although she is not technically a co-founder, she helps with a ton of admin work and, more importantly, has helped me stay positive. She has been as much a part of this startup as I have, and has suffered through the same things. If you're not with somebody, it might be harder or easier, depending on where you are in life. I know lots of single people in their 20s building successful companies on their own.

At the end of the day, it comes down to how you answer two questions: are you ever going to quit and can you adapt? You may not know this before starting, but if you do it long enough, you will have to figure out the answer. I've had to do this over the last year myself, and I've found the answer. (Hint: I'm still here.)

I could probably simplify this even further. The degree to which you are successful is a function of your ability to mold reality to your needs. This is something I learn a little more about the longer I'm in the game, because the work involved in developing that ability is significant.

In my opinion, #2 (having more hands to do the work) is the least important, although it is probably the one that most people think of as being hardest when they hear about people doing startups on their own. You could have 20 people on your team and there will always be more work than you can handle at any time.

Photo: Gopal Vijayaraghavan, *http://www.flickr.com/photos/t3rmin4t0r/3923451419/*.

> **“Any capable person on a small team wants to contribute to its success and it’s very easy for people to create busy work far too early.”**

In many early stage startups, having too many people can be a kiss of death (not that 2-4 is too many, necessarily.)

Any capable person on a small team wants to contribute to its success and it’s very easy for people to create busy work (or worse, spend money) far too early. When a company is trying to figure out what customers will pay for, it can be a handicap to build too much without knowing the answer to that question. We’ve learned from people like DHH the value of constraints and, from this perspective, there are few more constrained environments than a single person team trying to bootstrap a company into profitability.

Finally, #3 (a richer source of new ideas) may be even more important than #1 (emotional support). It is possible to engage in a variety of human relationships in order to get the support you need. But ideas (aside from leads) are the lifeblood of the entrepreneur. I know that it is fashionable these days to say that ideas are worth nothing and that it’s all about the execution. I don’t agree with that statement because it puts the focus on the wrong part of the idea. It may be true that each individual idea is worth less than the execution of that idea (although they are never worthless), what is really valuable is the ability to generate ideas. A surprising number of people just draw blanks when faced with challenges and those people who can deliver creative, new ideas about any given situation are pure gold.

In general, as the saying goes, two (or more) heads are better than one. So how does a solo founder get better at coming up with new ideas? I personally do it by trying learn about as many businesses as I can. I am fascinated by business models. I’ve done quite a bit of B2B sales over the last few years, selling the earliest versions of Ginzametrics. In my sales meetings, I always try to understand the intricacies of the customer’s business, down to the small details of what makes it tick. Almost always, I walk out of those meetings with more ideas of my own than I ever would have otherwise (and the sales process tends to go better to the degree that I understand their businesses.) There are many ways of getting new ideas. Talk to other startups. Read more books. Do little side projects that are completely unrelated to your product. I personally never feel at a loss for ideas (and have more than I know what to do with).

The last thing I want to write about is an observation I’ve made. Think about all of the successful companies we know. Isn’t it true in many cases (though not all, of course) that even in companies that were started by co-founders, there is usually The One?

The One is that person who really makes the company work. The other founders no doubt contribute a great deal, but if it really came down to it, the company would survive and flourish in much the same way as long as The One was running it. In some cases, this actually happens. Evan Williams, when he was working on Blogger, was reduced to being a single founder for awhile when his co-founders split. He is the reason that company worked. In the case of Mint.com, I believe Aaron Patzer actually was a solo founder and he is certainly the person that everyone in the Valley talks

> **“It’s not for everyone, but if you have an idea and want to do a startup but find yourself alone, just start doing it.”**

about as the one who made Mint.com work. Whenever I meet other startups, I’m always trying to figure out how they work. Is there one person in the company who really makes it work, or are they really a symbiotic team? In my own experience, it turns out to be half and half. Many of the startups I meet would work just fine with just The One, because they have that right mixture of charisma, determination and product vision.

This is important to note because, even if you start out on your own, it doesn’t mean you won’t someday be able to hire employees and recruit strong players for your executive team. Being a single founder is just a starting place. If you decide to raise money or achieve profitability on your own, you can hire the people you need to come up with new ideas and do more work. You’ve already demonstrated to yourself that you have or can find the emotional fortitude to survive. So what else do you need? It’s not for everyone, but if you have an idea and want to do a startup but find yourself alone, just start doing it. You’ll know soon enough if you’re cut out for it, and if you are, it won’t matter how many people you have on your team. ■

Ray Grieselhuber is the founder of Ginzametrics, which provides an easy way for companies to manage and improve their search engine optimization (SEO) campaigns. He blogs about his experiences and observations as an analytics startup at *http://ginzametrics.com/blog.html*.

Reprinted with permission of the original author. First appeared in *http://hn.my/singlefounder/*.

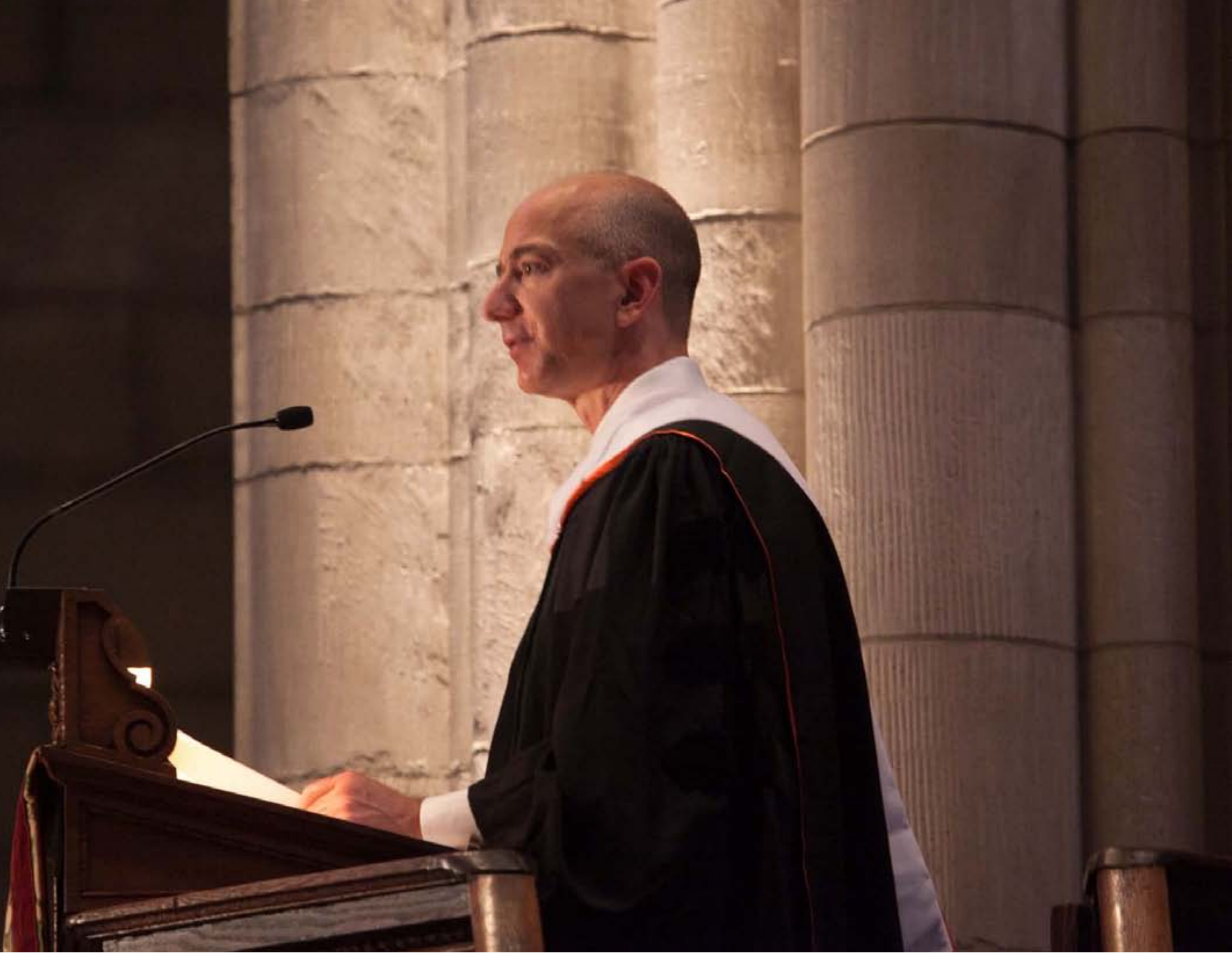
We Are What We Choose

Remarks by Jeff Bezos, as delivered to the Princeton Class of 2010
Princeton University Baccalaureate
May 30, 2010

As a kid, I spent my summers with my grandparents on their ranch in Texas. I helped fix windmills, vaccinate cattle, and do other chores. We also watched soap operas every afternoon, especially "Days of our Lives." My grandparents belonged to a Caravan Club, a group of Airstream trailer owners who travel together around the U.S. and Canada. And every few summers, we'd join the caravan. We'd hitch up the Airstream trailer to my grandfather's car, and off we'd go, in a line with 300 other Airstream adventurers. I loved and worshipped my grandparents and I really looked forward to these trips. On one particular trip, I was about 10 years old. I was rolling around in the big bench seat in the back of the car. My grandfather was driving. And my grandmother had the passenger seat. She smoked throughout these trips, and I hated the smell.

At that age, I'd take any excuse to make estimates and do minor arithmetic. I'd calculate our gas mileage – figure out useless statistics on things like grocery spending. I'd been hearing an ad campaign about smoking. I can't remember the details, but basically the ad said, every puff of a cigarette takes some number of minutes off of your life: I think it might have been two minutes per puff. At any rate, I decided to do the math for my grandmother. I estimated the number of cigarettes per days, estimated the number of puffs per cigarette and so on. When I was satisfied that I'd come up with a reasonable number, I poked my head into the front of the car, tapped my grandmother on the shoulder, and proudly proclaimed, "At two minutes per puff, you've taken nine years off your life!"

I have a vivid memory of what happened, and it was not what I expected. I expected to be applauded for my cleverness and arithmetic skills. "Jeff, you're so smart. You had to have made some tricky estimates, figure out the number of minutes in a year and do some division." That's not what happened. Instead, my grandmother burst into tears. I sat in the backseat and did not know what to do. While my grandmother sat crying, my grandfather, who had been driving in silence, pulled over onto the shoulder of the highway. He got out of the car and came around and opened my door and waited for me to follow. Was I in trouble? My grandfather was a highly intelligent, quiet man. He had never said a harsh word to me, and maybe this was to be the first time? Or maybe he would ask that I get back in the car and apologize to my grandmother. I had no experience in this realm with my grandparents and no way to gauge what the consequences might be. We stopped beside the trailer. My grandfather looked at me, and after a bit of silence, he gently and calmly said, "Jeff, one day you'll understand that it's harder to be kind than clever."



Jeff Bezos, speaking to the class of 2010 at Princeton's Baccalaureate service May 30.

What I want to talk to you about today is the difference between gifts and choices. Cleverness is a gift, kindness is a choice. Gifts are easy – they're given after all. Choices can be hard. You can seduce yourself with your gifts if you're not careful, and if you do, it'll probably be to the detriment of your choices.

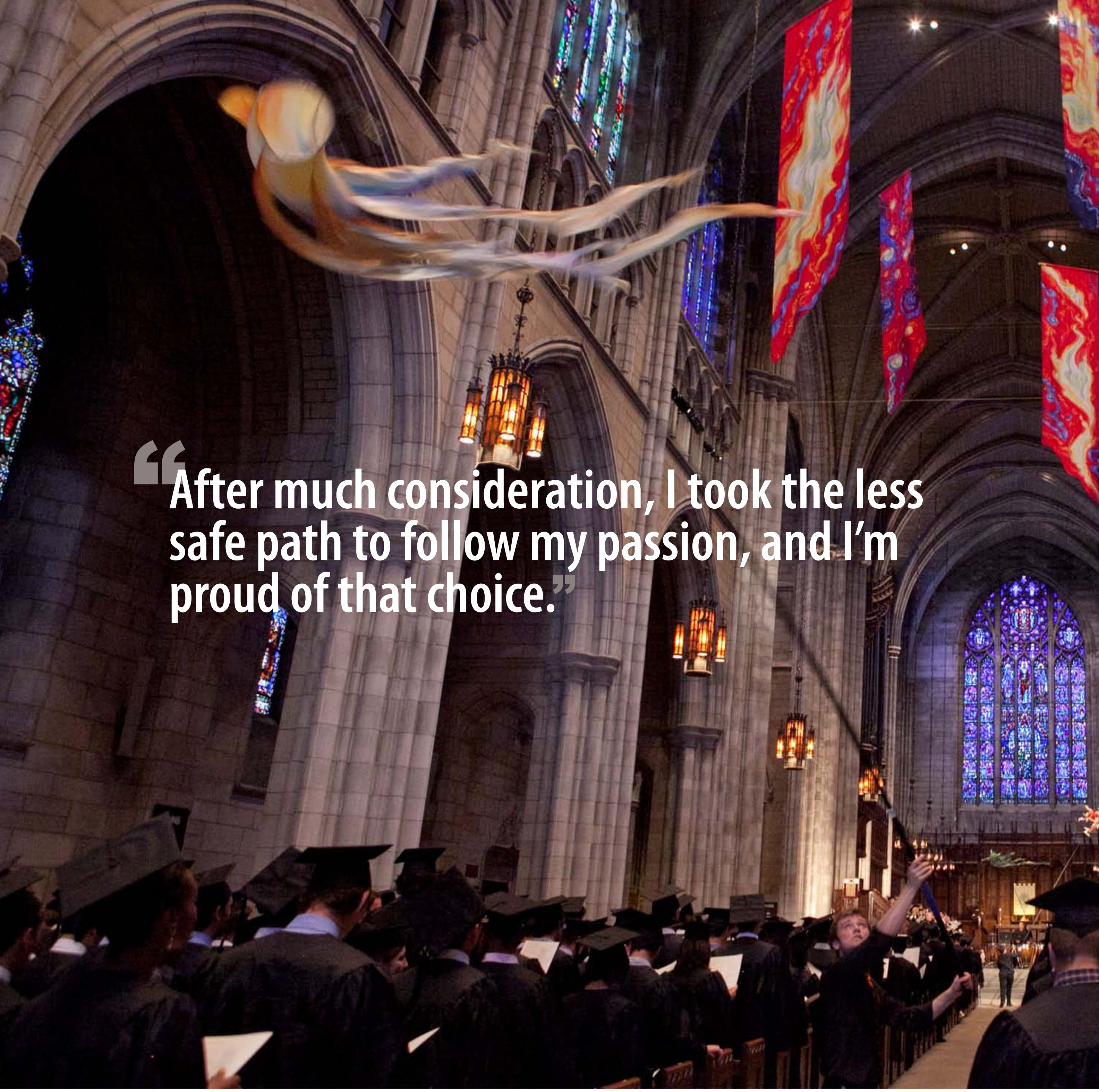
This is a group with many gifts. I'm sure one of your gifts is the gift of a smart and capable brain. I'm confident that's the case because admission is competitive and if there weren't some signs that you're clever, the dean of admission wouldn't have let you in.

Your smarts will come in handy because you will travel in a land of marvels. We humans – plodding as we are – will astonish ourselves. We'll invent ways to generate clean energy and a lot of it. Atom by atom, we'll assemble tiny machines that will enter cell walls and make repairs. This month comes the extraordinary but also inevitable news that we've synthesized life. In the coming years, we'll not only synthesize it, but we'll engineer it to specifications. I believe you'll even see us understand the human brain. Jules Verne, Mark Twain, Galileo, Newton – all the curious from the ages would have wanted to be alive most of all right now. As a civilization, we will have so many gifts, just as you as individuals have so many individual gifts as you sit before me.

How will you use these gifts? And will you take pride in your gifts or pride in your choices?

I got the idea to start Amazon 16 years ago. I came across the fact that Web usage was growing at 2,300 percent per year. I'd never seen or heard of anything that grew that fast, and the idea of building an online bookstore with millions of titles – something that simply couldn't exist in the physical world – was very exciting to me. I had just turned 30 years old, and I'd been married for a year. I told my wife MacKenzie that I wanted to quit my job and go do this crazy thing that probably wouldn't work since most startups don't, and I wasn't sure what would happen after that. MacKenzie (also a Princeton grad and sitting here in the second row) told me I should go for it. As a young boy, I'd been a garage inventor. I'd invented an

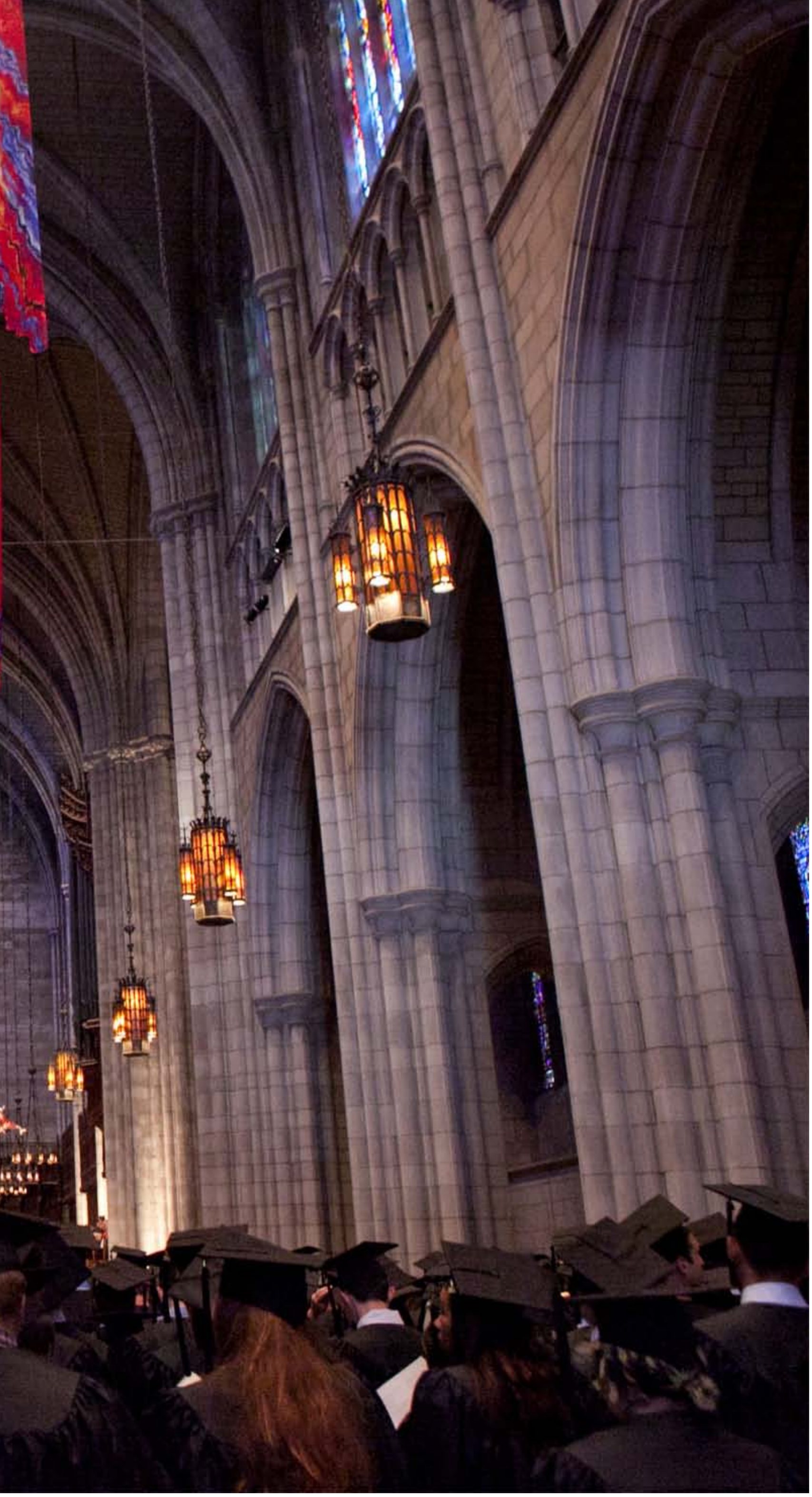
Photo: Princeton University, Office of Communications, Denise Applewhite



> “After much consideration, I took the less safe path to follow my passion, and I’m proud of that choice.”

automatic gate closer out of cement-filled tires, a solar cooker that didn’t work very well out of an umbrella and tinfoil, baking-pan alarms to entrap my siblings. I’d always wanted to be an inventor, and she wanted me to follow my passion.

I was working at a financial firm in New York City with a bunch of very smart people, and I had a brilliant boss that I much admired. I went to my boss and told him I wanted to start a company selling books on the Internet. He took me on a long walk in Central Park, listened carefully to me, and finally said, “That sounds like a really good idea, but it would be an even better idea for someone who didn’t already have a good job.” That logic made some sense to me, and he convinced



me to think about it for 48 hours before making a final decision. Seen in that light, it really was a difficult choice, but ultimately, I decided I had to give it a shot. I didn't think I'd regret trying and failing. And I suspected I would always be haunted by a decision to not try at all. After much consideration, I took the less safe path to follow my passion, and I'm proud of that choice.

TOMORROW, IN A very real sense, your life — the life you author from scratch on your own — begins.

How will you use your gifts? What choices will you make?

Will inertia be your guide, or will you follow your passions?

Will you follow dogma, or will you be original?

Will you choose a life of ease, or a life of service and adventure?

Will you wilt under criticism, or will you follow your convictions?

Will you bluff it out when you're wrong, or will you apologize?

Will you guard your heart against rejection, or will you act when you fall in love?

Will you play it safe, or will you be a little bit swashbuckling?

When it's tough, will you give up, or will you be relentless?

Will you be a cynic, or will you be a builder?

Will you be clever at the expense of others, or will you be kind?

I will hazard a prediction. When you are 80 years old, and in a quiet moment of reflection narrating for only yourself the most personal version of your life story, the telling that will be most compact and meaningful will be the series of choices you have made. In the end, we are our choices. Build yourself a great story. Thank you and good luck! ■

Reprinted with permission. First appeared in: http://www.princeton.edu/main/news/archive/S27/52/51O99/index.xml.
Photo: Princeton University, Office of Communications, Denise Applewhite

Plain English Explanation Of Big O Notation

By WILLIAM SHIELDS

I recently read A Beginners' Guide to Big O Notation and while I appreciate such efforts I don't think it went far enough. I'm a *huge* fan of "plain English" explanations to, well, anything. Just look at the formal definition of Big O. The only people who can understand that already know what it means (and probably have a higher degree in mathematics and/or computer science).

On StackOverflow you often get comments like "you should do X because it's O(2n) and Y is O(3n)". Such statements originate from a basic misunderstanding of what Big O is and how to apply it. The material in this post is basically a rehash and expansion of what I've previously written on the subject.

What is Big O?

Big O notation seeks to describe the *relative complexity* of an algorithm by reducing the growth rate to the key factors when the key factor tends towards infinity. For this reason, you will often hear the phrase *asymptotic complexity*. In doing so, all other factors are ignored. It is a **relative representation of complexity**.

What Isn't Big O?

Big O *isn't* a performance test of an algorithm. It is also *notional* or *abstract* in that it tends to ignore other factors. Sorting algorithm complexity is typically reduced to the number of elements being sorted as being the key factor. This is fine but it doesn't take into account issues such as:

- **Memory Usage:** one algorithm might use much more memory than another. Depending on the situation this could be anything from completely irrelevant to critical;
- **Cost of Comparison:** It may be that comparing elements is really expensive, which will potentially change any real-world comparison between algorithms;
- **Cost of Moving Elements:** copying elements is typically cheap but this isn't necessarily the case;
- etc.

Arithmetic

The best example of Big-O I can think of is doing arithmetic. Take two numbers (123456 and 789012). The basic arithmetic operations we learnt in school were:

- addition;
- subtraction;
- multiplication; and
- division.

Each of these is an operation or a problem. A method of solving these is called an **algorithm**.

Addition is the simplest. You line the numbers up (to the right) and add the digits in a column writing the last number of that addition in the result. The 'tens' part of that number is carried over to the next column.

Let's assume that the addition of these numbers is the most expensive operation in this algorithm. It stands to reason that to add these two numbers together we have to add together 6 digits (and possibly carry a 7th). If we add two 100 digit numbers together we have to do 100 additions. If we add two 10,000 digit numbers we have to do 10,000 additions.

See the pattern? The **complexity** (being the number of operations) is directly proportional to the number of digits. We call this **O(n)** or **linear complexity**. Some argue that this is in fact **O(log n)** or ***logarithmic complexity***. Why? Because adding 10,000,000 to itself takes twice as long as adding 1,000 to itself as there are 8 digits instead of 4. But 10,000,000 is 10,000 times as large so depending on your application it may be appropriate to define the problem in terms of number of digits (ie order of magnitude) of the input. In others, the number itself may be appropriate.

Subtraction is similar (except you may need to borrow instead of carry).

Multiplication is different. You line the numbers up, take the first digit in the bottom number and multiply it in turn against each digit in the top number and so on through each digit. So to multiply our two 6 digit numbers we must do 36 multiplications. We may need to do as many as 10 or 11 column adds to get the end result too.

If we have 2 100 digit numbers we need to do 10,000 multiplications and 200 adds. For two one million digit numbers we need to do one trillion (10^{12}) multiplications and two million adds. As the algorithm scales with n-*squared*, this is **O(n^2)** or **quadratic complexity**. This is a good time to introduce another important concept:

We only care about the most significant portion of complexity.

The astute may have realized that we could express the number of operations as: n2 + 2n. But as you saw from our example with two numbers of a million digits apiece, the second term (2n) becomes insignificant (accounting for 0.00002% of the total operations by that stage).

The Telephone Book

The next best example I can think of is the telephone book, normally called the White Pages or similar but it'll vary from country to country. But I'm talking about the one that lists people by surname and then initials or first name, possibly address and then telephone numbers.

Now if you were instructing a computer to look up the phone number for "John Smith", what would you do? Ignoring the fact that you could guess how far in the S's started (let's assume you can't), what would you do?

A typical implementation might be to open up to the middle, take the 500,000th and compare it to "Smith". If it happens to be "Smith, John", we just got real lucky. Far more likely is that "John Smith" will be before or after that name. If it's after we then divide the last half of the phone book in half and repeat. If it's before then we divide the first half of the phone book in half and repeat. And so on.

This is called a **bisection search** and is used every day in programming whether you realize it or not.

So if you want to find a name in a phone book of a million names you can actually find any name by doing this at most 21 or so times (I might be off by 1). In comparing search algorithms we decide that this comparison is our 'n'.

For a phone book of 3 names it takes 2 comparisons (at most). For 7 it takes at most 3. For 15 it takes 4. ... For 1,000,000 it takes 21 or so.

That is staggeringly good isn't it? In Big-O terms this is **O(log n)** or **logarithmic complexity**. Now the logarithm in question could be ln (base e), $\log_{10}$, $\log_2$ or some other base. It doesn't matter it's still O(log n) just like O($2n^2$) and O($100n^2$) are still both O(n^2).

It's worthwhile at this point to explain that Big O can be used to determine three cases with an algorithm:

- **Best Case:** In the telephone book search, the best case is that we find the name in one comparison. This is **O(1)** or **constant complexity**;
- **Expected Case:** As discussed above this is O(log n); and
- **Worst Case:** This is also O(log n).

Normally we don't care about the best case. We're interested in the expected and worst case. Sometimes one or the other of these will be more important.

Back to the telephone book.

What if you have a phone number and want to find a name? The police have a reverse phone book but such lookups are denied to the general public. Or are they? Technically you can reverse lookup a number in an ordinary phone book. How?

You start at the first name and compare the number. If it's a match, great, if not, you move on to the next. You have to do it this way because the phone book is **unordered** (by phone number anyway).

So to find a name:

- **Best Case:** O(1);
- **Expected Case:** O(n) (for 500,000); and
- **Worst Case:** O(n) (for 1,000,000).

The Travelling Salesman

This is quite a famous problem in computer science and deserves a mention. In this problem you have N towns. Each of those towns is linked to 1 or more other towns by a road of a certain distance. The Travelling Salesman problem is to find the shortest tour that visits every town.

Sounds simple? Think again.

If you have 3 towns A, B and C with roads between all pairs then you could go:

```
A -> B -> C
A -> C -> B
B -> C -> A
B -> A -> C
C -> A -> B
C -> B -> A
```

Well actually there's less than that because some of these are equivalent (A -> B -> C and C -> B -> A are equivalent, for example, because they use the same roads, just in reverse).

In actuality there are 3 possibilities.

Take this to 4 towns and you have (iirc) 12 possibilities. With 5 it's 60. 6 becomes 360.

This is a function of a mathematical operation called a factorial. Basically:

5! = 5 * 4 * 3 * 2 * 1 - 120
6! = 6 * 5 * 4 * 3 * 2 * 1 = 720
7! = 7 * 6 * 5 * 4 * 3 * 2 * 1 = 5040
...
25! = 25 * 24 * ... * 2 * 1 = 15,511,210,043,330,985,984,000,000
...
50! = 50 * 49 * ... * 2 * 1 = 3.04140932... × 10^64

So far, the only way known of solving the Travelling Salesman problem is by brute force. Unfortunately, such a technique has O(n!) complexity to solve.

By the time you get to 200 towns there isn't enough time left in the universe to solve the problem with traditional computers.

Something to think about.

Polynomial Time

Another point I wanted to make quick mention of is that any algorithm that has a complexity of **O(n^k)** for any constant *k* is said to have **polynomial complexity** or is solvable in **polynomial time**.

Traditional computers can solve problems in polynomial time. Certain things are used in the world because of this. Public Key Cryptography is a prime example. It is computationally hard to find two prime factors of a very large number. If it wasn't, we couldn't use the public key systems we use.

Big Greek Letters

Big O is often misused. Big O or Big Oh is actually short for Big Omicron. It represents the *upper bound* of asymptotic complexity. So if an algorithm is O(n log n) there exists a constant *c* such that the upper bound is *c*n log n.

Θ(n log n) (Big Theta) is more tightly bound than that. Such an algorithm means there exists two constants c_1 and c_2 such that c_1n log n < *f*(n) < c_2n log n.

Ω(n log n) (Big Omega) says that the algorithm has a *lower bound* of *c*n log n.

There are others but these are the most common and Big O is the most common of all. Such a distinction is *typically* unimportant but it is worth noting. The correct notation is the correct notation, after all.

Determinism

Algorithms can also be classified as being either **deterministic** or **probabilistic**.It's important to understand the difference. Sometimes requirements or constraints may dictate the choice of one over the other even if the expected case is worse. You should be able to classify an algorithm as one or the other.

A good example of this is comparing files. Say you have two files A and B and you want to determine if they are the same. The simple implementation for this is:

1. If the sizes are different, the files are different; else
2. Compare each file byte-for-byte. If two different bytes are found, the files are different; else
3. The files are the same.

This is a **deterministic** algorithm because the probability of a false positive (the algorithm saying the files are the same when they aren't) and a false negative (saying they are different when they aren't) is 0 in both cases.

For various reasons however it might be impractical or undesirable to implement the algorithm this way. Many file comparisons may be required making the operation potentially very expensive on large files. Also the files might be remote to each other and it might be impractical to send a complete copy just so the remote system can see if its changed.

A more common approach is to use a **hash function**. A hash function basically just converts a large piece of data into a smaller piece of data (called a **hash**), usually a 32-128 bit integer. A good hash function will distribute values in the new (smaller) data range as evenly as possible.

Reprinted with permission of the original author. First appeared in *http://hn.my/bigo/*.

A common hash function is an MD5 hash, which generates a 128-bit hash. Let's say files A and B were on different servers. One could send an MD5 hash of the file to the other, which could compare it to its own MD5 hash. If they're different, the files are different. If they're the same, the files are *highly likely to be the same*.

An MD5 hash comparison is a **probabilistic** comparison algorithm for this reason.

And before you say that the chance is so remote it'll never happen, *think again*. A malicious exploit has been demonstrated of generating two files with the same MD5 hash.

Algorithms such as this that only have brute force approaches age relatively quickly. Where once MD5 was considered safe, creating two messages with the same MD5 hash is now feasible (in a matter of days with not unreasonable hardware) such that the more secure SHA-1 algorithm has largely replaced it's usage.

Conclusion

Anyway, that's it for my (hopefully plain English) explanation of Big O. I intend to follow this up with applications to some common scenarios in weeks to come. ■

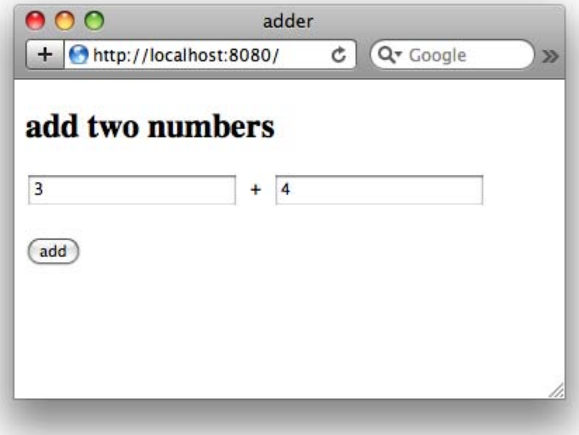
William Shields is a software developer from Perth, Western Australia who has been programming for over 20 years in Java, C/C++, Python, JavaScript and may things in between for financial trading and Web-based applications. He is a regular contributor to StackOverflow as cletus and blogs at *http://www.cforcoding.com*.

Developing And Deploying A Simple Clojure Web Application

By MARK MCGRANAGHAN

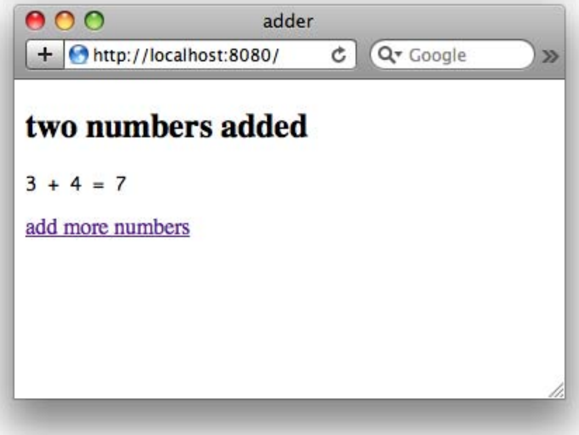
The post walks through the process of developing and deploying a simple web application in Clojure. After reading this you should be able to build your own app and deploy it to a production server.

Our sample app performs addition for the user. The user enters a value in each of two text fields, the values are submitted to the app, and the app returns the corresponding sum. Eventually it will look like this:



add two numbers

3 + 4



two numbers added

3 + 4 = 7

add more numbers

Before beginning on the app, make sure that you have Leiningen installed.

We'll start with a minimal first version of the app. In a new directory `adder`, create a file `project.clj` with the following contents:

```
(defproject adder "0.0.1"
  :description "Add two numbers."
  :dependencies
    [[org.clojure/clojure "1.2.0-beta1"]
     [org.clojure/clojure-contrib "1.2.0-beta1"]
     [ring/ring-core "0.2.5"]
     [ring/ring-devel "0.2.5"]
     [ring/ring-jetty-adapter "0.2.5"]
     [compojure "0.4.0"]
     [hiccup "0.2.6"]]
  :dev-dependencies
    [[lein-run "1.0.0-SNAPSHOT"]])
```

We'll put the main app logic in the namespace `adder.core`. Create a file at `src/adder/core.clj` with this code:

```
(ns adder.core
  (:use compojure.core)
  (:use hiccup.core)
  (:use hiccup.page-helpers))

(defn view-layout [& content]
  (html
    (doctype :xhtml-strict)
    (xhtml-tag "en"
      [:head
        [:meta {:http-equiv "Content-type"
                :content "text/html; charset=utf-8"}]
        [:title "adder"]]
      [:body content])))
```

```
(defn view-input []
  (view-layout
    [:h2 "add two numbers"]
    [:form {:method "post" :action "/"}
      [:input.math {:type "text" :name "a"}] [:span.math "
+ "]
      [:input.math {:type "text" :name "b"}] [:br]
      [:input.action {:type "submit" :value "add"}]]))

(defn view-output [a b sum]
  (view-layout
    [:h2 "two numbers added"]
    [:p.math a " + " b " = " sum]
    [:a.action {:href "/"} "add more numbers"]))

(defn parse-input [a b]
  [(Integer/parseInt a) (Integer/parseInt b)])

(defroutes app
  (GET "/" []
    (view-input))

  (POST "/" [a b]
    (let [[a b] (parse-input a b)
          sum   (+ a b)]
      (view-output a b sum))))
```

Also, put the following in `script/run.clj`:

```
(use 'ring.adapter.jetty)
(require 'adder.core)

(run-jetty #'adder.core/app {:port 8080})
```

Now you're ready to test the first version of the app:

```
lein deps
lein run script/run.clj
open http://localhost:8080/
```

Check out your app in the browser. You should be to perform the simple addition described above.

As you use the app you'll probably notice changes that you'd like to make. You might also notice that errors like giving `foo` as an input are not handled well. To fix this let's apply some reloading and stacktrace middleware.

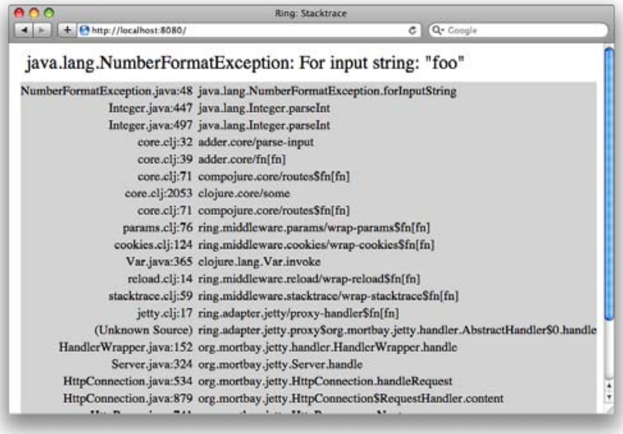
Start by including the appropriate Ring middlewares into the `adder.core` namespace definition:

```
(:use ring.middleware.reload)
(:use ring.middleware.stacktrace)
```

We'll want to separate out the main app logic that we wrote earlier from the full, middleware wrapped application, so change `(defroutes app` to `(defroutes handler` and add the following at the bottom of the file:

```
(def app
  (-> #'handler
    (wrap-reload '[adder.core])
    (wrap-stacktrace)))
```

After stopping your running server and restarting it with `lein run script/run.clj`, you should be able to see changes to your code in `adder.core` reflected immediately in the web interface. Also, if your application encounters any errors you will see a stacktrace indicating what went wrong:



java.lang.NumberFormatException: For input string: "foo"

```
NumberFormatException.java:48 java.lang.NumberFormatException.forInputString
             Integer.java:447 java.lang.Integer.parseInt
             Integer.java:497 java.lang.Integer.parseInt
                  core.clj:32 adder.core/parse-input
                  core.clj:39 adder.core/fn[fn]
                  core.clj:71 compojure.core/routes$fn[fn]
                core.clj:2053 clojure.core/some
                  core.clj:71 compojure.core/routes$fn[fn]
                params.clj:76 ring.middleware.params/wrap-params$fn[fn]
              cookies.clj:124 ring.middleware.cookies/wrap-cookies$fn[fn]
                 Var.java:365 clojure.lang.Var.invoke
                reload.clj:14 ring.middleware.reload/wrap-reload$fn[fn]
            stacktrace.clj:59 ring.middleware.stacktrace/wrap-stacktrace$fn[fn]
                 jetty.clj:17 ring.adapter.jetty/proxy-handler$fn[fn]
             (Unknown Source) ring.adapter.jetty.proxy$org.mortbay.jetty.handler.AbstractHandler$0.handle
      HandlerWrapper.java:152 org.mortbay.jetty.handler.HandlerWrapper.handle
              Server.java:324 org.mortbay.jetty.Server.handle
      HttpConnection.java:534 org.mortbay.jetty.HttpConnection.handleRequest
      HttpConnection.java:879 org.mortbay.jetty.HttpConnection$RequestHandler.content
```

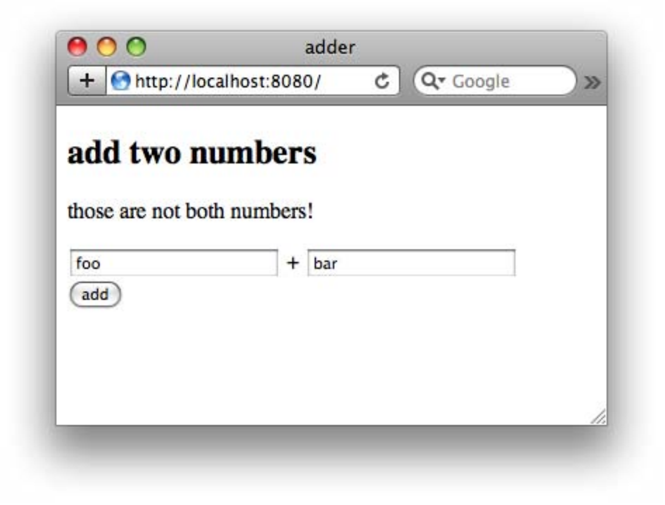
Speaking of errors, we may want to address some of those in our application. If a user enters something other than a number into one of the fields, we should respond with a useful error message. Update the view-input function to:

```
(defn view-input [& [a b]]
  (view-layout
    [:h2 "add two numbers"]
    [:form {:method "post" :action "/"}
      (if (and a b)
        [:p "those are not both numbers!"])
      [:input.math {:type "text" :name "a" :value a}]
[:span.math " + "]
      [:input.math {:type "text" :name "b" :value b}] [:br]
      [:input.action {:type "submit" :value "add"}]]))
```

and update the `POST` route to:

```
(POST "/" [a b]
  (try
    (let [[a b] (parse-input a b)
          sum   (+ a b)]
      (view-output a b sum))
    (catch NumberFormatException e
      (view-input a b))))
```

You can immediately verify that your changes worked by trying some invalid input:



add two numbers

those are not both numbers!

We should also handle the case where the user enters an unrecognized URL. To do that, require the Ring response middleware with:

```
(:use ring.util.response)
```

and then add a catchall route to the bottom of the routes list:

```
(ANY "/*" [path]
  (redirect "/"))
```

Now when you visit e.g. `/foo`, you should be redirected back to the app's main page at `/`.

Our app is starting to shape up, but we're missing some necessary application infrastructure. For one, the application is not doing any logging, which makes it hard to understand what it is doing. Lets fix that with some request logging middleware. Create a new file `src/adder/middleware.clj` with these contents:

```
(ns adder.middleware)

(defn- log [msg & vals]
  (let [line (apply format msg vals)]
    (locking System/out (println line))))

(defn wrap-request-logging [handler]
  (fn [{:keys [request-method uri] :as req}]
    (let [start  (System/currentTimeMillis)
          resp   (handler req)
          finish (System/currentTimeMillis)
          total  (- finish start)]
      (log "request %s %s (%dms)" request-method uri total)
      resp)))
```

Then pull this middleware into core with:

```
(:use adder.middleware)
```

and add it to the app by updating the middleware stack to look like:

```
(def app
  (-> #'handler
    (wrap-request-logging)
    (wrap-reload '[adder.middleware adder.
core])
    (wrap-stacktrace)))
```

Now each request will be noted in the app's logs, along with the time it takes.

As soon as you try out the logging you'll probably notice requests to `/favicon.ico`. Since our simple app doesn't have a favicon, let's let the browser know with a `404` response. Add a `wrap-bounce-favicon` function to the `adder.middleware` namespace:

```
(defn wrap-bounce-favicon [handler]
  (fn [req]
    (if (= [:get "/favicon.ico"] [(:request-method req)
(:uri req)])
      {:status 404
       :headers {}
       :body ""}
      (handler req))))
```

and then include it in the middleware stack by adding `(wrap-bounce-favicon)` immediately above `(wrap-stacktrace)`.

Now let's add a bit of styling to our utilitarian app. To do this we'll create and apply a CSS file that is served statically by the application. Put the following in public/adder.css:

```
.math {
  font-family: Monaco, monospace; }

.action {
  margin-top: 2em; }
```

and update the :head markup in view-layout to look like:

```
[:head
  [:meta {:http-equiv "Content-type"
          :content "text/html; charset=utf-8"}]
  [:title "adder"]
  [:link {:href "/adder.css" :rel "stylesheet" :type "text/
css"}]]
```

Next, include the necessary Ring middleware:

```
  (:use ring.middleware.file)
  (:use ring.middleware.file-info)
```

and update the middleware stack to look like:

```
(def app
  (-> #'handler
    (wrap-file "public")
    (wrap-file-info)
    (wrap-request-logging)
    (wrap-reload '[adder.middleware adder.
core])
    (wrap-bounce-favicon)
    (wrap-stacktrace)))
```

We should also write a few tests for our newly developed application. Create a file at `test/adder/core_test.clj` with the following contents:

```
(ns adder.core-test
  (:use clojure.test)
  (:use adder.core))

(deftest parse-input-valid
  (is (= [1 2] (parse-input "1" "2"))))

(deftest parse-input-invalid
  (is (thrown? NumberFormatException
    (parse-input "foo" "bar"))))

(deftest view-output-valid
  (let [html (view-output 1 2 3)]
    (is (re-find #"two numbers added" html))))

(deftest handle-input-valid
  (let [resp (handler {:uri "/" :request-method :get})]
    (is (= 200 (:status resp)))
    (is (re-find #"add two numbers" (:body resp)))))

(deftest handle-add-valid
  (let [resp (handler {:uri "/" :request-method :post
                       :params {"a" "1" "b" "2"}})]
    (is (= 200 (:status resp)))
    (is (re-find #"1 \+ 2 = 3" (:body resp)))))

(deftest handle-add-invalid
  (let [resp (handler {:uri "/" :request-method :post
                       :params {"a" "foo" "b" "bar"}})]
    (is (= 200 (:status resp)))
    (is (re-find #"those are not both numbers" (:body
resp)))))

(deftest handle-catchall
  (let [resp (handler {:uri "/foo" :request-method :get})]
    (is (= 302 (:status resp)))
    (is (= "/" (get-in resp [:headers "Location"])))))
```

You can verify that they all pass by running lein test.

Now that we have some tests we're ready to start thinking about deploying this app to production. We'll want the app to behave slightly differently in production and development, so we'll need a way to differentiate between the two environments. I'll use the environment variable APP_ENV to define production? and development? vars in the adder.core namespace:

```
(def production?
  (= "production" (get (System/getenv) "APP_ENV")))

(def development?
  (not production?))
```

Use this var to update the middleware stack to look like:

```
(def app
  (-> #'handler
    (wrap-file "public")
    (wrap-file-info)
    (wrap-request-logging)
    (wrap-if development? wrap-reload '[adder.middleware
adder.core])
    (wrap-bounce-favicon)
    (wrap-exception-logging)
    (wrap-if production?  wrap-failsafe)
    (wrap-if development? wrap-stacktrace)))
```

This code will enable a public-facing failsafe middleware in production while keeping the stacktrace middleware in development. We'll also limit code reloading to development. Finally, we'll add exception logging in both cases for additional visibility. This updated stack relies on several new functions in `adder.middleware`. Add the following to the `adder.middleware` namespace declaration:

```
(:require [clj-stacktrace.repl :as strp])
```

and to the `adder.middleware` body:

```
(defn wrap-if [handler pred wrapper & args]
  (if pred
    (apply wrapper handler args)
    handler))

(defn wrap-exception-logging [handler]
  (fn [req]
    (try
      (handler req)
      (catch Exception e
        (log "Exception:\n%s" (strp/pst-str e))
        (throw e)))))

(defn wrap-failsafe [handler]
  (fn [req]
    (try
      (handler req)
      (catch Exception e
        {:status 500
         :headers {"Content-Type" "text/plain"}
         :body "We're sorry, something went wrong."}))))
```

The site will not run on port `8080` in production, so we'll need a way to specify the port to the run script. We'll use the `PORT` environment variable. Update the body of `script/run.clj` to the following:

```
(let [port (Integer/parseInt (get (System/getenv) "PORT"
"8080"))]
  (run-jetty #'adder.core/app {:port port}))
```

Now we're ready to put this app into production. I'll walk through the steps needed for to deploying to EC2 using the standard EC2 command line tools, but the process would be similar for other hosting providers.

Start be allocating by setting up a security group and SSH keypair for the application:

```
ec2-add-group adder -d "adder deployment"
ec2-authorize adder -P tcp -p 22
ec2-authorize adder -P tcp -p 80

mkdir -p dev
ec2-add-keypair adder | tail -n +2 > dev/adder.pem
chmod 600 dev/adder.pem
```

Then allocate a server based on a public Ubunut AMI and wait for it to come up:

```
ec2-run-instances ami-2d4aa444 -g adder -k adder \
  -n 1 -t m1.small -z us-east-1a
watch ec2-describe-instances
```

Set some local environment variables to make subsequent commands easier:

```
export ADDER_PEM=dev/adder.pem
export ADDER_HOST=<ec2-public-ip>
```

To set up the server, SSH in

```
ssh -i $ADDER_PEM ubuntu@$ADDER_HOST
```

and run a few commands to install Java and set up the directory structure:

```
sudo su root
curl -L -o install-java.sh http://bit.ly/b5lesP
bash install-java.sh
mkdir -p /var/log/adder /var/adder
chown -R ubuntu:ubuntu /var/adder
```

We'll control the server process using Ubuntu's upstart. Put the following upstart configuration file in `deploy/adder.conf`:

```
script
  export PORT=80
  export APP_ENV=production
  cd /var/adder
  java -cp "lib/*:src/" clojure.main script/run.clj \
    >> /var/log/adder/adder.log 2>&1
end script
```

and then place it in the appropriate spot on the server with:

```
scp -i $ADDER_PEM deploy/adder.conf \
  ubuntu@$ADDER_HOST:/tmp/adder.conf
ssh -i $ADDER_PEM ubuntu@$ADDER_HOST \
  "sudo mv /tmp/adder.conf /etc/init/adder.conf"
```

Reprinted with permission of the original author. First appeared in *http://hn.my/clojure/*.

Create a list in `deploy/exclude.txt` of files that should not be deployed to the production server:

```
.git
.gitignore
deploy
test
classes
```

Now install the app's files on the server with:

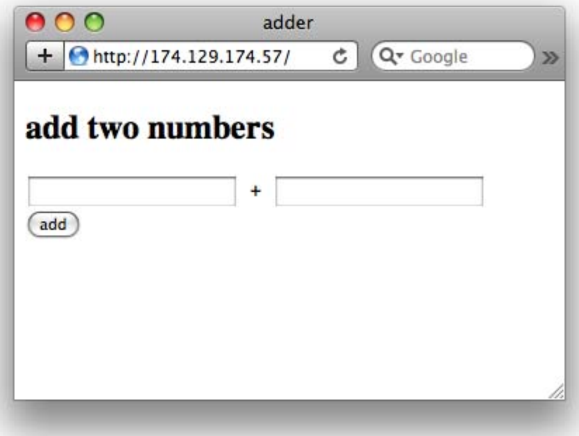
```
rsync --rsh='ssh -i '$ADDER_PEM \
      -vr --delete --exclude-from deploy/exclude.txt \
      ./ ubuntu@$ADDER_HOST:/var/adder/
```

After the first rsync completes, start the server with:

```
ssh -i $ADDER_PEM ubuntu@$ADDER_HOST "sudo start adder"
```

and check that it works by opening the production site from your local machine:

```
open http://$ADDER_HOST/
```



If you want to deploy a change, `rsync` up your code and then run:

```
ssh -i $ADDER_PEM ubuntu@$ADDER_HOST "sudo restart adder"
```

I hope this post helps you develop and deploy your own Clojure web applications. If you have any questions about this post or about Clojure web development in general, feel feel to leave them in the comments. I'm also interested in hearing how others have approached the end-to-end Clojure web development and deployment process; please let me know what you think in in the comments as well.

The source code for this app is available on *http://github.com/mmcgrana/adder*. ■

Mark McGranaghan is an engineer at Heroku, where he works on web application infrastructure. He maintains several open source Clojure projects at *http://github.com/mmcgrana*.

Criminal Overengineering

By MARK O’CONNOR

AS PROGRAMMERS WE’RE continually accused of doing a sloppy job. There are countless programs in the wild, crashing, locking up and accidentally writing “I am a fish” a million times over someone’s mid-term essay. The effect? Something like this:

> ***This damn computer and excel r fuckin my life up! Hatin life right now***
> – MissAlauren (and everyone else at one time or another)

It’s experiences like this that cause people to rant about Microsoft and curse the anonymous programmers who suddenly (yet inevitably) betrayed them. We all know this; it’s burned into our very souls by countless hours of tech support provided to family and friends. Time after time we see that programmers who do quick, sloppy work make other people suffer. And so we try, we try so damn hard not to be like that. We try to be the good programmer who checks every return value and handles every exception.

If we stopped at competent error handling and sufficient testing, all would be well. In truth, we actually go too far and, it has to be said, largely in the wrong direction.

A vast proportion of software at work today is horribly over-engineered for its task. And I’m not talking about the interfaces, about having too many controls or options for the users. These are, indeed, terrible sins but they are the visible ones. The worst of the overengineering goes on under the surface, in the code itself.

You’re Doing It Wrong

Have you ever seen someone using the strategy pattern when they should’ve used a 5 line switch statement? There are a million ways to turn something like this:

```
switch(operation)
{
    case OP_ADD: return a + b;
    case OP_SUBTRACT: return a - b;
    case OP_MULTIPLY: return a * b;
    default: throw new UnknownOperationException(operation,
a, b);
}
```

… into a hideous, malformed mutant beast like this one *http://en.wikipedia.org/wiki/Strategy_pattern#Java*, which I haven’t inlined because it’s far too long.

The most insidious cause of overengineering is over-generalizing. We will over-generalize anything given half a chance. Writing code to work with a list of students? Well, we might want to work with teachers and the general public someday, better add a base People class and subclass Student from that. Or Person and then EducationPerson and then Student. Yes, that’s better, right?

Only, now we have three classes to maintain each with their own virtual methods and interfaces and probably split across three different files plus the one we were working in when a one-line dictionary would have been fine.

Perhaps we do it because it’s relaxing to rattle off three classes worth of code without needing to pause and think. It feels productive. It looks solid, bulletproof, professional. We look back on it with a comforting little glow of self-satisfaction – we’re a good programmer, no messy hacks in our code.

Except, this doesn’t make us good programmers. Overengineering like this isn’t making anyone’s lives better; it’s just making our code longer, more difficult to read and work with and more likely to contain or develop bugs. We just made the world a slightly worse place. It lies somewhere between tossing an empty drinks bottle on the street and grand theft auto.

The extra effort caused by our overengineering carries a hefty opportunity cost:

1. Less time spent refining the user experience
2. Less time spent thinking about the meaningful implications of the feature we're working on
3. Less time available to look for bugs and – with harder-to-read code – more time spent debugging them

Yes, by overengineering the Student class you indirectly ruined MissAlauren's day.

We have to stop championing each ridiculous feat of overengineering and call it what it is. It's not 'future-proof', because we can't see the future. It's not robust, it's hard to read. Applying a generic solution to a single case isn't good programming, it's criminal overengineering because like it or not somebody, somewhere will pay for it.

Don't Worry, Be Happy

I suspect all the best programmers have already realized this, but they're not shouting about it loudly enough for everyone else to hear. Paul Graham is completely right when he suggests that succinctness is valuable:

> *Use the length of the program as an approximation for how much work it is to write. Not the length in characters, of course, but the length in distinct syntactic elements – basically, the size of the parse tree. It may not be quite true that the shortest program is the least work to write, but it's close enough… look at a program and ask, is there any way to write this that's shorter?*
> – Paul Graham, The Hundred Year Language

He's actually talking about language design here; indeed, in Succinctness is Power he's careful to note that it's clearly possible to write a program that's too succinct. This is because, these days, Paul Graham is more a language designer than a working programmer. Otherwise he might have said:

> *If you're about to take a hundred lines to write what you could in ten, stop and ask yourself this: what the fuck?*
> – Mark, Criminal Overengineering

When I feel tempted to over-generalize or over-engineer a bit of code, it's often because of fear. Fear that someone will find a really good reason I shouldn't have done it the easy way. Fear that I'll have to rewrite the code again. Fear of finding myself on the wrong side of an argument about the merits of the visitor pattern. But fear does not naturally lead us to the most elegant solutions.

Next time you feel the compulsion to write a nice, general solution to a simple case, stop and ask yourself what's stopping you just writing it the simple, specific, succinct way:

1. Am I worried I'll have to rewrite it?
2. Am I worried someone will criticize it or that I'll look bad?
3. Am I worried that it's not professional enough?

Are any of these true? Then relax. Don't worry. You worry, you call me, I make you happy.

Just write the code the simple, specific way and then add a short comment, something like: Replace with the Strategy pattern if this gets any bigger.

This is the perfect solution. It's a great reminder to you next time you come here about what you wanted to do. It shows other programmers on your team that you considered the 'correct' way to do it and have a good reason not to do it just yet. It's very hard to argue with a comment like that, because you're not arguing about the strategy pattern vs the switch statement, you're arguing about whether to use the strategy pattern after 3 cases or after 4 cases – not a discussion that can reflect badly on you, in any case.

A few months later you can go back and look at how many of your comments eventually turn into more complicated, engineering code. I'll bet you it's not very many. That's how much time and effort you've saved, right there. That's setting yourself free to pursue the solution and that's making the world a slightly better place. ■

Mark O'Connor is a programmer, occasional writer and part-time startup founder. Since 2008 he's been living in Munich with his wife and children. He believes in dynamic typing, first-class functions and the immortal essence of the human soul. He also likes tea.

Reprinted with permission of the original author. First appeared in *http://hn.my/overengineering/*.

Experimenting With nodeJS

By JEFF KREEFTMEIJER

This is an experiment I did to play around with Node.js and web sockets. I've put everything in a Gist in case you want to try it out yourself. I'll explain how it works in this article.

Web socket server

Using @miksago's node-websocket-server made it extremely easy to send and receive messages from a web socket. Here's the code that runs the server:

```
var ws = require(__dirname + '/lib/ws'),
    server = ws.createServer();

server.addListener("connection", function(conn){
  conn.addListener("message", function(message){
    message = JSON.parse(message);
    message['id'] = conn.id
    conn.broadcast(JSON.stringify(message));
  });
});

server.addListener("close", function(conn){
  conn.broadcast(JSON.stringify({'id': conn.id, 'action':
'close'}));
});

server.listen(8000);
```

After including the node-websocket-server library and creating the server, I add some listeners to know when clients disconnect or send a message and make sure messages get sent to the other clients. Whenever it receives a JSON message, it includes the connection's id before broadcasting it to the clients to make it possible to find out which cursor we need to move.

I saved it as server.js, so starting the server is as simple as running node server.js. To make sure it keeps running, I daemonized it with God, using the same config file I used in the "Daemonizing Navvy with God" article.

Receiving messages

Now, in a regular javascript file — with some jQuery — I included into this page, I connect to the web socket like this:

```
var conn;
var connect = function() {
  if (window["WebSocket"]) {
    conn = new WebSocket("ws://jeffkreeftmeijer.com:8000");
    conn.onmessage = function(evt) {
      data = JSON.parse(evt.data);
      if(data['action'] == 'close'){
        $('#mouse_'+data['id']).remove();
      } else if(data['action'] == 'move'){
        move(data);
      };
    };
  }
};

window.onload = connect;
```

As you can see, this connects to the server we just started. When a message is received, it checks the action it's supposed to perform. If the action is "move", it'll move a mouse cursor on the screen using the move() function I'll show you later. If it's "close", it means that the client disconnected and his cursor has to be removed from the screen.

Sending messages

Now we're able to receive messages, move and delete cursors. The last thing we need is the client to be able to send out messages:

```
$(document).mousemove(
  ratelimit(function(e){
    if (conn) {
      conn.send(JSON.stringify({
        'action': 'move',
        'x': e.pageX,
        'y': e.pageY,
        'w': $(window).width(),
        'h': $(window).height()
      }));
    }
  }, 40)
);
```

Whenever you move your mouse, the .mousemouse() function gets triggered that sends some JSON with the mouse position and screen size to the socket. The ratelimit method makes sure that there's a forty millisecond interval between messages.

Moving the cursors

So, when the other clients receive a "move" message, it calls the move() function, like I showed you before. It looks like this:

```
function move(mouse){
  if($('#mouse_'+mouse['id']).length == 0) {
    $('body').append(
      '<div class="mouse" id="mouse_'+mouse['id']+'"/>'
    );
  }

  $('#mouse_'+mouse['id']).css({
    'left' : (($(window).width() - mouse['w']) / 2 +
mouse['x']) + 'px',
    'top' : mouse['y'] + 'px'
  })
```

It creates a div for the new mouse if it doesn't exist yet and moves it to the right position. Also, the x-position of the mouse gets calculated while keeping the difference in screen size in mind. This way it gets calculated from the center of the page, instead of from the left.

Blew your mind?

Tracking mouse movement and showing cursors to other clients is cool, but not useful in any way (although you could think of some cool use-cases for this). What this example does show is that you can do pretty impressive things using web sockets and Node.js. And it was a great excuse to play around with it.

This was the first thing I did using Node.js, so the code is probably far from perfect. If you know a way to improve it, please fork the Gist and show me how it should be done. I'll update the article.

I'm excited about Node.js and I'll probably write and play around with it some more in the future, so stay tuned. ■

Notes

1. Gist at *http://gist.github.com/488562*
2. Live demo - *http://hn.my/nodejs/*
3. If you see extra mouse cursors moving around: don't worry, they're part of the demo. You can always disable them if you want. These are actually other people also looking at the page right now, live, as we speak. If you don't see anything, try to open up this page in another browser window and move your mouse in it.
4. I've written a follow-up on this article, in which I improved a lot of the code. Be sure to read that one too! *http://hn.my/nodejs2/*

Jeff Kreeftmeijer is an open source enthusiast and Ruby (on Rails) programmer at 80beans in Amsterdam. He publishes weekly programming articles on *jeffkreeftmeijer.com*, writing about Ruby or anything else that seems interesting. As of late, that has mostly been JavaScript and Node.js.

Reprinted with permission of the original author. First appeared in *http://hn.my/nodejs/*.

The Absolute Bare Minimum Every Programmer Should Know About Regular Expressions

By MIKE MALONE

Regular expressions are strings formatted using a special pattern notation that allow you to describe and parse text. Many programmers (even some good ones) disregard regular expressions as line noise, and it's a damned shame because they come in handy so often. Once you've gotten the hang of them, regular expressions can be used to solve countless real world problems.

Regular expressions work a lot like the filename "globs" in Windows or *nix that allow you to specify a number of files by using the special * or ? characters (oops, did I just use a glob while defining a glob?). But with regular expressions the special characters, or metacharacters, are far more expressive.

Like globs, regular expressions treat most characters as literal text. The regular expression mike, for example, will only match the letters m - i - k - e, in that order. But regular expressions sport an extensive set of metacharacters that make the simple glob look downright primitive.

Meet the metacharacters: ^[](){}.*?\|+$ and sometimes -

I know, they look intimidating, but they're really nice characters once you get to know them.

The Line Anchors: '^' and '$'

The '^' (caret) and '$' (dollar) metacharacters represent the start and end of a line of text, respectively. As I mentioned earlier, the regular expression mike will match the letters m - i - k - e, but it will match anywhere in a line (e.g. it would match "I'm mike", or even "carmike"). The '^' character is used to anchor the match to the start of the line, so ^mike would only find words that start with mike. Similarly, the expression mike$ would only find m - i - k - e at the end of a line (but would still match 'carmike').

If we combine the two line anchor characters we can search for lines of text that contain a specific sequence of characters. The expression ^mike$ will only match the word mike on a line by itself - nothing more, nothing less. Similarly the expression ^$ is useful for finding empty lines, where the beginning of the line is promptly followed by the end.

The Character Class: '[]'

Square brackets, called a character class, let you match any one of several characters. Suppose you want to match the word 'gray', but also want to find it if it was spelled 'grey'. A character class will allow you to match either. The regular expression gr[ea]y is interpreted as "g, followed by r, followed by either an e or an a, followed by y."

If you use [^ ...] instead of [...], the class matches any character that isn't listed. The leading ^ "negates" the list. Instead of listing all of the characters you want to included in the class, you list the characters you don't want included. Note that the ^ (caret) character used here has a different meaning when it's used outside of a character class, where it is used to match the beginning of a line.

The Character Class Metacharacter: '-'

Within a character-class, the character-class metacharacter '-' (dash) indicates a range of characters. Instead of [01234567890abcdefABCDEF] we can write [0-9a-fA-F]. How convenient. The dash is a metacharacter only within a character class, elsewhere it simply matches the normal dash character.

But wait, there's a catch. If a dash is the first character in a character class it is not considered a metacharacter (it can't possibly represent a range, since a range requires a beginning and an end), and will match a normal dash character. Similarly, the question mark and period are usually regex metacharacters, but not when they're inside a class (in the class [-0-9.?] the only special character is the dash between the 0 and 9).

Matching Any Character With a Dot: '.'

The '.' metacharacter (called a dot or point) is shorthand for a character class that matches any character. It's very convenient when you want to match any character at a particular position in a string. Once again, the dot metacharacter is not a metacharacter when it's inside of a character class. Are you beginning to see a pattern? The list of metacharacters is different inside and outside of a character class.

The Alternation Metacharacter: '|'

The '|' metacharacter, (pipe) means "or." It allows you to combine multiple expressions into a single expression that matches any of the individual ones. The subexpressions are called alternatives.

For example, Mike and Michael are separate expressions, but Mike|Michael is one expression that matches either.

Parenthesis can be used to limit the scope of the alternatives. I could shorten our previous expression that matched Mike or Michael with creative use of parenthesis. The expression Mi(ke|chael) matches the same thing. I probably would use the first expression in practice, however, as it is more legible and therefore more maintainable.

Matching Optional Items: '?'

The '?' metacharacter (question mark) means optional. It is placed after a character that is allowed, but not required, at a certain point in an expression. The question mark attaches only to the immediately preceding character.

If I wanted to match the english or american versions of the word 'flavor' I could use the regex flavou?r, which is interpreted as "f, followed by l, followed by a, followed by v, followed by o, followed by an optional u, followed by r."

The Other Quantifiers: '+' and '*'

Like the question mark, the '+' (plus) and '*' (star) metacharacters affect the number of times the preceding character can appear in the expression (with '?' the preceding character could appear 0 or 1 times). The metacharacter '+' matches one or more of the immediately preceding item, while '*' matches any number of the preceding item, including 0.

If I was trying to determine the score of a soccer match by counting the number of times the announcer said the word 'goal' in a transcript, I might use the regular expression go+al, which would match 'goal', as well as 'goooooooooooooooooal' (but not 'gal').

The three metacharacters, question mark, plus, and star are called quantifiers because they influence the quantity of the item they're attached to.

The Interval Quantifier: '{}'

The '{min, max}' metasequence allows you to specify the number of times a particular item can match by providing your own minimum and maximum. The regex go{1,5}al would limit our previous example to matching between one and five o's. The sequence {0,1} is identical to a question mark.

The Escape Character: '\'

The '\' metacharacter (backslash) is used to escape metacharacters that have special meaning so you can match them in patterns. For example, if you would like to match the '?' or '\' characters, you can precede them with a backslash, which removes their meaning: '\?' or '\\'.

When used before a non-metacharacter a backslash can have a different meaning depending on the flavor of regular expression you're using. For perl compatible regular expressions (PCREs) you can check out the perldoc page for perl regular expressions. PCREs are extremely common, this flavor of regexes can be used in PHP, Ruby, and ECMAScript/Javascript, and many other languages.

Using Parenthesis for Matching: '()'

Most regular expression tools will allow you to capture a particular subset of an expression with parenthesis. I could match the domain portion of a URL by using an expression like http://([^/]+). Let's break this expression down into it's components to see how it works.

The beginning of the expression is fairly straightforward: it matches the sequence "h - t - t - p - : - / - /". This initial sequence is followed by parenthesis, which are used to capture the characters that match the subexpression they surround. In this case the subexpression is '[^/]+', which matches any character except for '/' one or more times. For a URL like http://immike.net/blog/Some-blog-post, 'immike.net' will be captured by the parenthesis.

Want to know more?

I've only touched the surface on what can be done with regular expressions. If want to learn more, check out Jeffrey Friedl's book Mastering Regular Expressions. Friedl did an outstanding job with this book, it's accessible and a surprisingly fun and interesting read, considering the dry subject matter. ■

Mike Malone is an infrastructure engineer at SimpleGeo where he works on building and integrating scalable systems that power the company's location platform. In his spare time Mike enjoys tinkering with new technologies. When he's not on the computer, you can probably find him hanging out with his girlfriend, Katie, and their friends at a good bar.

Reprinted with permission of the original author. First appeared in *http://hn.my/regex/*.

On: Developing And Deploying A Simple Clojure Web Application

From MAHMUD MOHAMED (mahmud)
In Common Lisp (without cheating or handwaving; this is it):

```
  (defpackage :add-nums
    (:use :cl :hunchentoot :cl-who))

  (in-package :add-nums)

  (defmacro with-html (&body body)
    `(with-html-output-to-string (*standard-output* nil
:prologue t :indent t)
       ,@body))

  (define-easy-handler (add-nums :uri "/add-nums")
      ((a :parameter-type 'integer)
       (b :parameter-type 'integer))
    (with-html
      (:html
       (:head (:title "Add two numbers"))
       (:body
        (if (and a b)
          (htm (:p (str (+ a b)))))
        (:form :method :get
         (:input :type :text :name "a")
         (:input :type :text :name "b")
         (:input :type :submit))))))

  (defparameter *site* (make-instance 'acceptor :port
8080))

  (start *site*)
```

On: Criminal Overengineering

From PETER ARONOFF (telemachos)
Argh.

Apparently there are two very popular types of article in the software blog world:

Type 1: YAGNI (like this example): Do less now. Refactor later as needed. It won't be needed, most likely. Chill out. (All driven by the question, "Dude, wtf? 100 lines of boilerplate for a 5 line case statement? Snap out of it.")

Type 2: Architect astronautics: Do more now. Build for the next version. You will need more then, so why not prepare now? Decouple that code. Use more patterns. Hoist that jib. (All driven by the question, "What will your code/software/app do if...?")

I read type 1, and it (often) sounds convincing. I read type 2, and it (often) sounds convincing. I get a fucking headache from the cognitive dissonance. I make more coffee and get back to work, no wiser than before.

On: If You Can't Buy Your Investor A Beer, Don't Take Their Money

From MARK MAUNDER (mmaunder)
"If you need money, you are too late." really means "If you appear to need money, you are too late." And restated the way one of my investors once put it: "If you have the stench of death about you it's impossible to raise."

If you don't need money, don't raise.

On: A Coder's Guide To Coffee

From JOHN FORSYTHE (chaosmachine)
If you find "mainstream" coffee enjoyable, is it really in your best interest to raise your expectations? If you suddenly find normal coffee lacking, have you really improved your life?

On: How To Focus

From DOUG TOLTON (NyxWulf)
I used to use techniques like this to maintain my focus. Let me toss something out though that might be a bit controversial. Perhaps having to do this is a sign of ADHD. For me it was. There are many signs and symptoms, and unfortunately many people have heard that people are over medicated on things like Ritalin and Adderral. However, the thing to consider is that people with strong ADHD actually have a different brain structure. It turns out that the focus benefits of being on the medication only works for people who genuinely do have ADHD, and does not work for "regular" people (if there even is such a thing).

Let me just say, that Adderral changed my life. I used to have what I termed "Reddit seeking behavior". I could work on tasks that were interesting, but if it got a little rough or boring - what's happening on Reddit, or HN or Digg? However when I got on Adderral that entirely changed. I still check HN, but much less frequently because I'm busy getting things done. I don't have to resort to extreme changes in my environment to resist distractions. It comes naturally now.

I don't discount or disagree that focus can be improved. I know it first hand, but I would also argue that people with ADHD have some additional things going on that are much harder to train. If you are one of those people, getting on the appropriate medication could well change your life. I know it changed mine. That's a big statement, and often over-used, but for me, it changed the way I experience every day life in substantial and dramatic ways.

Take it for what it's worth, YMMV.

On: The Royal We: Single Founder Startups

From PATRICK MCKENZIE (patio11)

I think that the (increasing but by no means new) viability of single-founder startups is an inevitable consequence of the environment for all startups continuing to get radically better. If two guys could make non-trivial web services back when making a web service started with, quite literally, programming your own HTTP server since you didn't have a hundred thousand to buy one, that implies very good things for a "team" which has a decade and change of OSS to lean on.

In addition to OSS, the huge existing distribution channels like organic SEO, AdWords, and all those things you cool people use are also a major draw. Infrastructure has improved by orders of magnitude. APIs and snap-in services are getting better all the time — ten years ago, payment processing was a multi-week endeavor, now you can do voice calls in about ten minutes of work. Scaling is... is solved too strong a word? There has been huge diffusion of the black magic of how to setup and architect things, both in the n-tier server architecture sense of the word and in the "here's how you get capital without slicing open chicken entrails" and "here's how you get users" senses of the word.

It is a great time to be alive.

On: Three Years Ago, I Sold My Startup Because I Was An Idiot

From JACQUES MATTHEIJ (jacquesm)

I'm going to take the contrary view here and I'll say that you did fine.

Starting a business, operating it for a while and a successful exit are the whole meal, if you had continued to run it you might have crashed it or you might have been in debt now.

There is no way to know what would have happened in an alternative universe, and it's a pretty good thing on your resume to say that you ran a startup that made you and your investors money. That puts you in the < 5% bracket of the entrepreneurs.

Now you simply need to do that all again, with the lessons learned and your newfound energy, capital and the success story behind you.

You'll do very well indeed if you play your cards right.

Don't be so hard on yourself, don't compare with the 'could have been' from the perspective of it could have been more, it could have been a whole lot less too!

All comments are reprinted with permission of their original author.

TECH JOBS

Backend Engineer

Dropbox (*http://www.dropbox.com*)
San Francisco

Dropbox is looking for an engineer to scale our increasingly complicated backend. We're a multi-petabyte distributed filesystem with tens of billions of files serving millions. You will design and maintain core software components – using C/C++. You will have expertise navigating shell environments and significant experience with databases.

To Apply: *http://hn.my/dropbox1/*

Technical Support Engineer

Dropbox (*http://www.dropbox.com*)
San Francisco

Supporting Dropbox requires crazy technical kung-fu. You'll need a working knowledge of almost all operating systems, mobile devices, and the web. You'll be tasked with not only knowing the product inside and out, but escalating good suggestions and critical bugs straight to the product team. Most of our support is via email, forums and phone.

To Apply: *http://hn.my/dropbox2/*

Senior Developer

youDevise, Ltd. (*https://dev.youdevise.com*)
London, England

60-person agile financial software company in London committed to learning and quality (dojos, TDD, continuous integration, exploratory testing). Under 10 revenue-affecting production bugs last year. Release every 2 weeks. Mainly Java, also Groovy, Scala; no prior knowledge of any language needed.

To Apply: Send CV to *jobs@youdevise.com*.

cause
LATIN AMERICA
BizTechDay
MSL QUARTERLY
SPEAKING
Wild Life
YOUTH RUNNER
MAGCLOUD
Find Magazines