Staying Healthy and Sane At a Startup

By Alex Payne

HACKERMONTHLY Issue 6 November 2010

Curator
Lim Cheng Soon

Proofreader
Ricky de Laveaga

Illustrators
Jaime G. Wong
Matthew D. Phelan

Printer
MagCloud

E-book Conversion
Fifobooks.com

Contributors
ARTICLES
Zed A. Shaw
Alex Payne
Patrick Mckenzie
Cameron Chapman
Rahul Vohra
Marco Arment
Bronnie Ware
Darius A Monsef IV
Sebastian Marshall
ridiculous_fish
Bryan Hales
James Hague
Daniel Markham

COMMENTS
Reginald Braithwaite
Raphaël Amiard
Daniel Krol
Jacques Mattheij
Nir Yariv
Dave Gallagher
Tom Darrow

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*.

Advertising
ads@hackermonthly.com

Contact
contact@hackermonthly.com

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

Cover Illustration: Matthew D. Phelan

Contents

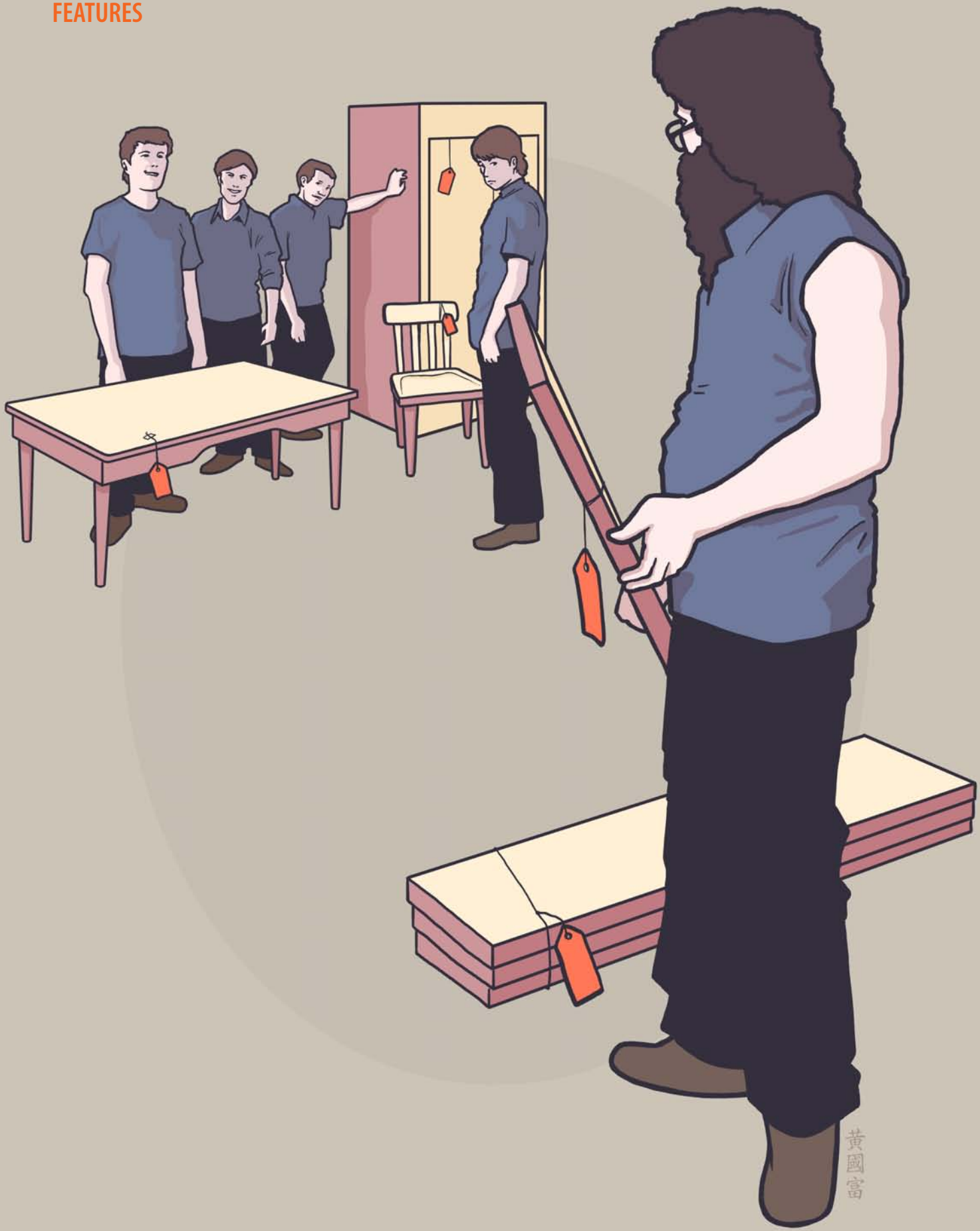
Illustration: Jaime G. Wong

FEATURES

STARTUPS

SPECIAL

PROGRAMMING


黃國富

Products For People Who Make Products For People

By ZED A. SHAW

I HAVE SEVERAL FRIENDS who are "Product People." Their self-proclaimed definition as a Product Person is that they focus on the "user experience" of everything they make, and could care less about the "backend." To them the entire purpose of software is a magic show whereby a programmer creates the perception of quality through graphic design regardless of the actual quality held within.

To the Product Person I am a dinosaur. I'm a Long Beard. I'm a guy who makes web servers for fun and gives them away. First, the fact that I actually know how to make a web server just boggles their mind. I might as well tell them I can make a Jeep Cherokee from raw iron ore. The reaction is about the same. Second, to them a web server isn't "product," it's infrastructure. It's not even a toilet, it's the rusty pipe that feeds water to the toilet.

To a Product Person the things I make are laughable. They aren't products because people don't use them, only programmers. To make a good web server you just have to code. There's no design, no usability, no human elements at all. The all superior Product(TM) has design, usability, and is used by humans. "Your web server is just used by geeks and it's just code."

The crux of the Product Person's belief system is this idea that unless the product has a graphic component then it's not a product and it has no elements of usability. And if it's not a product then it's looked down on as not worth their time. It's obviously a stupid idea, but where did it come from?

The Inmates Weren't Running The Asylum

I think this Product Person attitude can be traced directly to The Inmates Are Running The Asylum by Alan Cooper. It is one of many books that advocated the idea that programmers should be kept away from products and that business leaders are the ones who should be in charge. The book should have really been called "Fucking Nerds" because it was very abusive and simply categorically wrong in many ways.

The primary problem with the book (and really the entire belief that programmers screw up usability) is that it assumed they were in control. It assumed that programmers made the decisions in the product, which is just flat out wrong since most programmers already just do what some biz dude tells them. It also assumed programmers controlled the technologies they used, which again is plain wrong since they typically had to eat what a corporation like Microsoft fed them.

Illustration: Jaime G. Wong

> “If Long Beards are autistic then Product People are sociopaths.”

Let me put it into perspective this way. Let’s use the example of a Bank website. They’re horrible, so according to Alan Cooper’s view it’s the programmer’s fault. They’re really in control and should be completely removed from the product decision and turned into just factory workers creating exactly what the obviously more in-touch executives tell them to create. Programmers are at fault for the abysmal online banking experience.

Alright, people who think that obviously haven’t worked at a bank as a programmer. First off, you are told where you will work, what team you have, and given a strict process to follow. You are told you will use Microsoft products, or Sun products; and use Visual Studio, or Eclipse. You will use windows, and interface with an antiquated COBOL system. You are told that it must work with various departments because they have budget. You are told that you must have headcount of 16 people and buy 10 servers you don’t need. You are told that you will add “corn flower blue” to all the icons because some Big Swinging Dick said his daughter’s favorite color is cornflower blue.

You are told what to do with everything and typically have no say in the actual product in this situation. All you have control over is how you use the tools they’ve chosen for you. Alright, you can wield your code to make a better product right?

Nope, because the tools they’ve given you are again controlled by some corporation with a certain design ideal. If it’s Microsoft then the things you have to work with are Microsoft looking and feeling. If it’s Oracle they are Oracle looking and feeling. Stepping outside of those predefined tool chains is incredibly difficult, but if you do then someone above you yells at you to “keep it professional.” What they mean is, keep Microsoft’s nasty looking clip art just like everything else at the bank.

But, let’s say you can pull this off and you have permission to really make the UI sexy. Alright, where’s your designer? In every mega-corp and government agency I’ve worked for there has never been a staff designer of any kind. If there was one he or she was barely capable and totally out of touch with modern design. How can a programmer possibly make a good visual design without any help from a professional designer? That’s like making a designer code up the web server in C++ and then blaming them when whatever they make sucks.

Despite this truth that programmers have very little control, Alan Cooper’s book took off and spawned an entire generation of “Fucking Nerds” books. Every one of these books made the assumption (either explicit or implied) that if you could code hard core stuff like web servers then you couldn’t make a decent product. There was even an implied offensive insult that technical competence meant you had autism. You didn’t know people and it’s only the Product People who deserve the rewards and credit for anything, not nerds.

Incidentally, I’d say, if Long Beards are autistic then Product People are sociopaths. Just saying, the insults can go both ways.

The Inmates Created The Palace

Keep in mind that back when this book was written technology just barely worked for consumer products. The web was kind of crappy, desktop graphics were just barely there, and the tools to build better ones were defined mostly by a few companies with an Enterprise aesthetic. Trying to build a consumer product that looked sexy in those days was damn near impossible.

Here, take a look at Paypal and compare it to Heroku. Paypal looks like ass compared to Heroku, but back when Alan Cooper wrote his book, Paypal was the height of sexy product interfaces. Now it looks like junk compared to today’s graphic design, but today’s design is only possible because browsers got better and competitors to IE6 came out.

Paypal existed in the world where IE6’s horrible aesthetic and stagnant implementation ruled the world, and that’s why it looks that way. Not because “Nerds” screwed it up. Even still, Paypal is a very easy to use product and it makes a ton of money, so in a way it proves that good design isn’t really the only product consideration.

> **“The frontend product is what brings your revenue stream in, but your backend operations quality is what keeps your costs down as you grow.”**

What’s happened is that programmers who hated these tools spent their evenings and free time making better tools. They slaved away at better browsers, better languages, better graphics, better operating systems, all sorts of “backend” infrastructure that the current crop of Product People take for granted.

Without the same programmers that Alan Cooper ranted against in his book you would not have any of these products without massive capital investment. Guys like me also hated the way things looked, but there wasn’t much we could do about it because the tech just wasn’t there. Alan Cooper seemed to think that programmers just wave a wand and POOF there’s product, but the truth is we build our products off other products.

And, the products we used to build products just sucked horribly.

Product People Are Right And Wrong

Obviously Product People are right in that most programmers who make infrastructure software do make unusable crap. They’re also right in advocating the mantra that products be usable and that we need to focus on who the end user is, not on just the cool hack. They are right that programmers of my generation need to learn usability as well. This is important.

Where Product People go wrong is in two assumptions:

❶ Infrastructure software does not have usability concerns.
❷ The internal quality of a product doesn’t matter.

I’ll cover the first assumption later with what I know are the usability concerns of infrastructure software, but first let’s talk about the lack of real quality in most Product People products.

Obviously the internal quality of a product does matter, it’s just that current products that you can create on the internet let you hide crap internals. Since my time in San Francisco I’ve found many, many companies who seem to be making awesome high quality products when the truth is, internally, they’re piles of duct taped junk. This works for a while, but eventually they get bit in the ass when a competitor with better tech comes along and just copies them.

Or even worse, when the technical debt that comes from ignoring internal quality of a product creeps up on them and the costs destroy the company.

I like to think of the division of “frontend” vs “backend” to work a lot like this:

- Revenue - Cost = Profit
- Product - Operations = Profit
- Design - Implementation = Profit
- Frontend - Backend = Profit

Meaning, the frontend product is what brings your revenue stream in, but your backend operations quality is what keeps your costs down as you grow. If your backend costs get out of control because of technical debt then you won’t make a profit, or someone who can keep them down will just copy you and wipe you out with less. Pretty simple.

I believe that the current crop of “products” created by Product People are in for a big crash. They’ll eventually have such huge cost overruns that they’ll never turn a profit. There of course are complexities in that statement because of economies of scale in hosting, but all these cheap cloud services do is stave off the inevitable. If your product is entirely focused on the user experience (revenue stream) and not the operations (cost reduction) then you’ll have a hard time turning a real profit.

The Coming Long Beard Revolt

Even worse I think guys like me are gonna revolt. Younger coders tend to have no respect for older coders because of this idea that there’s nothing to learn from the Long Beards. In the past that was true because software development as a profession hadn’t really solidified. There weren’t a lot of really good programmers and the technology changed too fast.

I believe that we’re at an interesting point where the Long Beards are valuable because technology hasn’t changed all that much. What I see is in the previous technological revolutions, when the technology went away the programming languages backing them (and thus the idioms and knowledge) went with them. Mainframes died and so did COBOL.

In this latest set of technology shifts though, the programming languages being used have been plentiful, old, and adapted to the new landscape. Clojure is Lisp. Erlang is Prolog. Java is C++. Ruby is Smalltalk and Perl. Long beards won't necessarily be wiped out with each revolution anymore because there's a damn good chance their language(s) of choice are going to be used on the next one.

And if their languages or similar ones are being used again, then the Long Beard's knowledge and expertise does matter. Assuming they can get their head out of their ass and actually be bothered to learn the new stuff (that's just like the old stuff) they'll find that they have skills that Product People need to reduce costs.

But here's what I see coming. I see the Long Beards figuring out that they are needed and revolting. I see the revolt being a combination of:

❶ A serious dip in the amount of free stuff Product People need to survive.
❷ A sudden rise in Long Beards simply copying Product People Products but doing it cheaper using their cost reducing backend skills.

However, in order for that to happen I think the Long Beards need to learn how to make the special brand of products they make usable first. Once they figure out that their skills are operations level cost reducers, and learn who their real users are (Product People) then they'll be in a good position to dominate.

Because honestly, making a usable site isn't too hard if you have a little bit of cash. Hire a designer, read a few books, use a couple usability experiment techniques, etc. are all non-difficult things to learn. I think it takes about maybe a year or so to learn usability (notice I said usability, not design) if you actually care.

However, learning all the intricacies of high performance web servers, databases, and programming language design can take decades. I actually think in this faux competition between Long Beards and Product People the Long Beards have the advantage because their "products" are more fad resistant and their skills can translate to many more opportunities.

But First, you guys have to learn how to make...

Products For People Who Make Products For People

Infrastructure software obviously does have usability concerns, but the concerns are different because the people who use them are different. Where Long Beards go wrong is they seem to write software that's written for computers, not people. They sit down and write code, servers, and APIs that are just impossible to use or understand, and then don't document it. Then if someone tries to use it and has problems they assume that person is an idiot. Obviously if the original author can use the thing then everyone can, so anyone who can't must be a moron.

This is why people hate Long Beards. While Product People are hated because they come off as con artists hustling for an extra buck, Long Beards are hated because they seem to hate people.

In fact, Long Beards go so far as to irrationally think that if a piece of infrastructure software is easy to use then it's crap. If it has an entertaining manual, then Long Beards scoff at the "flowery language." If it has a minimal set of viable options they rant about it being too simple. If it adopts a vogue technology for some part they laugh at it being for "kids" or a "fad."

Really it is like they think people don't use their software, and partially make Alan Cooper's idiotic book right. This attitude is just as bad as Product People thinking software without graphics can't have usability. Because, Long Beards may not make software for people, but they do make software for people who make software for people.

People are still using your software. It doesn't matter if you do operating systems design, algorithm design, or write web servers. Your software is used by a real person at some point. Someone has to call your functions so if you name them weird your software is hard to use. Your operating system will be setup and used by a person, so if it is bizarre and missing key functions it is hard to use. Your web server is run and setup and managed by a person so if it had bad docs and obtuse error messages it is hard to use.

Your end users are Product People. You need to toss out this stupid idea that making something usable by DHH fanbois means you're not HARD CORE. You can still be hard core and make something they can use, hell something any programmer can use. By doing this you will reduce costs for the people who use your software which is what that kind of software is good at.

The shift in thinking is to focus on usability as if it ware a linguistic concern rather than a graphical concern. Product People focus on the design and interaction of their product through graphics because that's how their customers have to interact with what they make.

Long Beards need to focus on the design and interaction of their product through linguistics because that's how Product People interact with infrastructure software.

Usable Infrastructure Software

To make my software usable I focus on the linguistic elements of the design. The things I create don't really have a graphical component a person interacts with daily. What they interact with are software APIs, command line tools, configuration files, databases, build scripts, package managers, and automation tools to constrain all of those.

Linguistics are your user interface. Now this essay is already really long, so I won't go into all the finer points of making a great linguistic experience™® but here's some of the things I try do when making my stuff.

The most important thing is I rely very heavily on parsers as a core component of my software designs. The reason is a parser is the most reliable and proven way to safely and cleanly handle linguistic input. If you code up your config files, command line interfaces, protocol formats, and other linguistic elements by hand then it becomes very hard to explain them to your Product People users.

With a parser, you have a very succinct and clear explanation of the grammar

that they have to use. Parsers give better error messages, cleaner code, and have solid math backing them so there's much higher quality in general.

Parsers also force you to make your linguistic interfaces logical. If you're creating APIs you already are constrained by the parser in your chosen programming language. If you're doing other textual inputs then a parser's core design and mathematical basis forces you to make the language logical. It's just too hard to throw in weird warts if you stick to a good parser and solid grammar structure.

Parsers are the first line of a good linguistic experience, but they're not the only thing that make infrastructure software easy to use. They help, but you really need a set of other linguistic helpers:

❶ **Copious and clear error messages that explain both one problem and potentially how to fix it.** Make your errors psychic even. I have errors that predict failed branches and buffer overflows which end with "Tell Zed." People then come and tell me when they hit one. It's great. I also try to include the file:line location so people can hunt down exactly where the error is and possibly fix it.

❷ **Extensive and fun to read documentation.** Gone are the days of dry boring documentation. Your docs don't have to be Pulitzer worthy, but you have to give people something more than just a mind numbing stream of facts. Academic language is also out. Keep it conversational, full of information they need, and something they want to read.

❸ **Full support systems.** That includes code repositories, bug tracking, mailing lists, wiki systems, all the things your users need to go get information, get help, or report errors. These don't need to be really complex or entirely too open, but you do need something there.

❹ **Code that reads well.** I see too many programmers who write code that just doesn't read very well. Either because it's too complex, abuses too many concepts at once, or because of simple formatting choices. My personal pet peeve is people who don't add space to their code. The ENTER and SPACEBAR are free people, use them.

❺ **Assumption of reasonable defaults.** Don't make people specify every damn thing, just assume some basic defaults and let them change the defaults if they need to.

❻ **Reduce feature set by making clear choices.** Too often programmers try to include tons of features and then make the end user pick which ones to use. Instead I try to have a limited set of features, slowly add on as needed, and then create a module system for extensions.

❼ **Leverage familiar existing linguistic interfaces**, but don't repeat past mistakes. It's good to give people something they already know, but don't just copy something because everyone else does it. Branch out and do something better but with a bit of familiarity. Like how I gave people a config file system using a sqlite3 database, but crafted two different config file interfaces for it. Nobody needs to use SQL and can use a familiar interface, but they also don't have all the problems of a config file.

❽ **Test new features and design ideas by writing the docs for them first.** Typically if you can't explain the feature easily in writing then it's not designed well.

❾ **Automated testing.** Product People can get away with not having test automations like unit tests, test scripts, etc. Your stuff doesn't have the luxury of the magic show, so has to be tight and trust worthy, so testing counts.

❿ **Finally, less linguistic experience is better.** The more linguistics a person has to deal with the harder it is for them to get them right. A good metric is the size of your grammar in your parsers. If your grammar is approaching the size of a programming language then that's probably too much.

Now, these recommendations also have an additional purpose of reducing the cost of and time of running your project. If you have a hard time accepting that linguistic experience is your goal in infrastructure software, then at least think of these recommendations as a way to not have to deal with people who use your stuff.

The Boolean Is Just For Effect

One final thought is that, while I did make a division between Product People and Long Beards for effect, I don't actually think there's such a clear division. If you agreed with this division then I'd say you've got a problem and probably need to start experimenting with being the other type of person for a while. Everyone in tech is a product person and a long beard at the same time, it just depends on what you're doing.

I think the artificial division between the two can probably be summarized as:

> *The problem we have today is that Long Beards think focusing on product and usability means you're a flake hippie, and Product People think focusing on technical quality means you're an aspie neckbeard.*

The truth is if you want to be good at this stuff you've gotta be both in varying degrees at different times. If you irrationally force your identity into one of these stereotypes then you're not going to be as good as you could be. ■

Zed Shaw is the author of "Learn Python The Hard Way", many rants, essays, and has been blogging for as long as there's been blogging. He also created various web servers, email servers, and random open source projects, some of which actually power real companies. He is currently working on the Mongrel2 web server and in his spare time is obsessed with guitars.

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

Staying Healthy and Sane At a Startup

By ALEX PAYNE

I did a lot of things wrong while at Twitter. First and foremost, I took pretty terrible care of myself during our crazy early days (2007 – 2008). I'd had intermittently demanding jobs before, but nothing like the unrelenting stress and chaos of a fast-growing startup. I was a wreck for most of those two years, and I wasn't even working the insane hours of, amongst others, our head operations guy at the time.

During that time period, I was constantly getting sick. I had nothing resembling a consistent sleep schedule. I'd pile on weight from stress-eating, then burn it off from stress-not-eating. Relationships fell apart. My code was adequate, but I was scatterbrained, and I produced little that was up to the quality I expect from myself. Generally, it sucked. I sucked. And I promised never to let work get the best of me again.

Most sensible people take a vacation between jobs. That wasn't really an option for me when I left Twitter to join BankSimple earlier this summer. The company needed to raise its Series A (which we just closed), and I was too excited about getting started to sit around for half a month. But while I opted not to take a break, I knew that I'd have to change my habits in a big way in order to survive this time around.

Here's what I've been doing–or at least trying to do–to stay healthy and sane while working on a startup. It's not rocket science. It may work for you, and it may not. But these strategies have been helpful for me, so I thought I'd share, in hopes that others have an easier time of it.

Exercise

This is a no-brainer: get as much exercise as you possibly can. I try to exercise daily. I work out for three reasons: stress relief, energy, and long-term health. The last reason is self-explanatory, but the first two are worth explaining.

Startups are stressful. Exercise combats stress. Punchy meeting? Code that just won't do your bidding? Sweat it out. I'm not a naturally athletic person, and going to the gym is usually utterly unappealing after a long day. At the end of a good workout, though, I always feel calmer than when I started. Exercise boosts my mood and makes me more able to see negative or combative situations from a more positive perspective.

Startup life will sap your energy. At first, it's easy to operate on sheer enthusiasm. Over time, though, even the most exciting job becomes work. Working out can tire out the muscles, but I find that it energizes my mind. If I exercise regularly, I don't get antsy during the day. This lets me focus for longer periods on tasks that may not be thrilling but have to get done, like piles of paperwork or project planning.

Personally, I belong to a gym, and I do a mix of cardio (elliptical, stationary bike) and weight lifting, with some basic stretching on either end. I listen to podcasts while I work out to make the time go faster, and to sometimes learn something. Ninety minutes in the gym can feel like wasted time. Of course, maintaining one's health is far from a waste, but for geeks, time not spent working or learning usually feels squandered. Taking in a brainy podcast at the gym combats that feeling.

Later this month I'm moving away from my current neighborhood and, by extension, my current gym. I'm considering ditching a traditional gym for frequent CrossFit classes, and perhaps a return to Krav Maga, which I studied briefly years

ago and enjoyed. The more I've gotten into an exercise routine, the more it starts to feel, well, routine. Both CrossFit and a martial art have the promise of adding appealing variety, and of avoiding the dreaded "fitness plateau" (which I'm currently in no danger of reaching).

Point is: exercise. It works. It's the most straightforward of the recommendations I'm making here.

Diet

My metabolism sucks. My ancestry is primarily a mix of English and German, and as a result I'm genetically optimized for storing fat through a chilly European winter (also for arch looks and laconic humor). If I don't eat carefully, I gain weight, and if I gain weight, I look and feel like crap. Without strict rules about what I can and can't eat, I'll find myself eating whatever's around, particularly when I'm stressed from work. To combat this, I set very clear guidelines about what I eat and drink, and when.

Programmers notoriously live on caffeine and sugar. I refuse to cut the caffeine out of my diet, but the biggest change I've made for myself is cutting out refined sugar. Basically, the only "sweet" in my diet comes from fruit, or small quantities of chocolate. The only exception I make for sugar is in the occasional cocktail, but I've limited those, too (see below).

I've also removed most "bad" carbohydrates and starches from my diet. I avoid bread, pasta, white rice, potatoes, etc. So yes, that means no sandwiches, no noodles, no fries; none of a lot of things that I enjoy. These restrictions seem like more of an ordeal when I'm hungry, but by the time I'm done eating something that fits the guidelines I've set for myself, I'm no longer feeling deprived.

In essence, the diet I've ended up with is something akin to the South Beach Diet, but not taken to an extreme. I don't count calories, monitor the glycemic index of the foods I'm eating, or try to aggressively induce "phases" of weight loss. I just try to eat fresh vegetables, lean protein, low-fat dairy, nuts, and fresh fruit. This regime removes a huge number of readily available and hideously unhealthy foods as meal options. Being able to say, "nope, that's just not in the category of things that I eat" is helpful when confronted with a menu or grocery store full of choices.

I've gone a step further and restricted my alcohol intake to only days that don't precede work days. So, in a typical week, that means I only get to drink on Friday and Saturday. This has been the hardest dietary change for me to make. Anyone who follows me on Twitter knows that I love booze; not to get drunk, but just for the wonderful range of flavors and creativity exhibited in good beer, wine, spirits, and cocktails.

Though I miss my evening drink, this change has been worth it. Cutting out alcohol for most of the week means a huge savings in calories. Avoiding drinking before work days means that I'm fresh and ready to go in the morning. I've found that it's harder to get to the gym when I've had alcohol the previous night, so avoiding booze helps maintain my commitment to exercise.

The point of all these dietary changes is primarily about achieving constancy. Yes, it's nice to lose some weight, but by sticking to the above rules, my energy level throughout the day remains the same. Removing the sugar and carbs means that I don't peak and trough. I generally feel less ruled by food, and it's easier to make dietary decisions now that I have a framework.

Illustration: Matthew D. Phelan

Meditation

This is probably the most important of the changes I've made. Regular meditation is absolutely essential to maintaining quality of life for me. It keeps me calm and focused, and helps me sort out personal and professional conundrums.

The meditation technique I use is called Natural Stress Relief, or NSR. Yes, their site looks goofy and dated, and maybe even a bit sketchy, but have a Google around and you'll find out that NSR is reasonably well-known and accepted. It's dead simple to do: sit normally in a chair, clear your mind, silently repeat a monosyllabic mantra for about fifteen minutes, clear you mind again, and you're done. Repeat twice daily. I found it worthwhile to get the official PDF + MP3 guide on the technique, as in practice it's slightly more nuanced than my quick description, but thankfully not by much.

I chose NSR after doing some research on different techniques. There are many ways to meditate, and also many different goals to meditation. Being a devout agnostic, I'm not looking to commune with the spirits, become one with a deity, or reach enlightenment; I just want to feel like I've got my head screwed on straight. Most of the techniques out there are either derived from or actively grounded in religious practice, but not so with NSR. It's completely secular, and has no goal other than improving the mental state of the practitioner. I like the method's simplicity and its pragmatism.

The hardest part of meditation is making the time to do it. Realistically, you need about 20 minutes per NSR session. While that doesn't sound like much, adding 20 minutes to your morning and evening routines is harder than you think. It's entirely worth it, though. Meditation cuts right through feelings of being stressed-out and overwhelmed, and neatly organizes thoughts and emotions. More than once, I've been meditating and have had the solution to a problem I've been struggling with pop to the forefront of my mind. That's time well spent.

In a way, meditation is an investment in the quality of time spent not meditating. Even if you don't have any magic moments of clarity while sitting there with your eyes closed, you'll probably find that the rest of your day just feels better when you meditate regularly. At the very least, meditation makes my work time more productive, and that alone makes it worthwhile for me.

Time Management

I've always been reasonably well organized, but time management is distinct from organization. I've found that time management has little to do with "lifehacks" and how you manage your email inbox and more to do with prioritization, saying "no" to people, and clearly communicating the expectations you have for yourself and others. I'm less crazed this time around the startup block because I feel that I have a better grasp on how to manage my time, both during the workday and when I'm off the clock.

A big part of this shift was realizing that time spent in front of a desk isn't necessarily useful work time. If you're burned out for the day, stop working; go relax, exercise, or meditate, and come back to work with renewed energy and focus. That's an easy policy to get behind, but harder to put into practice, particularly in traditional office environments. American culture at large is no stranger to a Puritan work ethic, and that labor fanaticism is magnified all the more so in the startup "community" through legends of all-nighters and weeks spent sleeping under desks. Get over the guilt and bullshit, and realize that you'll be happier, healthier, and more productive if you manage work time on your terms.

This is probably the section where my advice is the least clear. From my perspective, time management is less a set of techniques than a mindset, albeit one assisted by social skills that allow you to defend your time and sanity. If you're totally new to the idea of time management, this talk by "last lecture" professor Randy Pausch will get you started. Once you're set with keeping a calendar, working through a task list, and batching your phone and email sessions, the broader mindset of time management is acquired through experience. You'll figure out what works for you, and where you need to draw boundaries.

Finally

Of course, everything in moderation, and all within reason and good taste. Though I'm trying to cut out sugar, I didn't turn down a slice of wedding cake at my friend's nuptials over the weekend. If I'm catching a 6AM flight, I'm probably going to miss my morning meditation session, and maybe miss that day's workout, too. I just try to keep the good habits going, and recover from lapses as quickly as possible.

I hope at least some of the above is helpful to someone. It goes without saying that everyone is different, and what works for me may be disastrous for you. But, if you're working on a startup or about to embark on one, I'd encourage taking the opportunity to examine your habits and see if you can't improve yourself as much as you're trying to improve the world around you. ■

Alex Payne is a cofounder of BankSimple, a startup combining modern technology with extraordinary customer service to create a seamless, worry-free banking experience. Previously, he was one of the first engineers at Twitter. Alex is the coauthor of "Programming Scala" (O'Reilly, 2009), and has been writing online for about a decade. He's a recent transplant to Portland, Oregon.

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

SEEING IS BELIEVING

RUBY ON RAILS 3 TUTORIAL SCREENCAST SERIES

by Michael Hartl, author of Rails Tutorial and RailsSpace

"My former company (CD Baby) was one of the first to loudly switch to Ruby on Rails, and then even more loudly switch back to PHP… This book by Michael Hartl came so highly recommended that I had to try it, and Ruby on Rails Tutorial is what I used to switch back to Rails again… Though I've worked my way through many Rails books, this is the one that finally made me 'get' it."

—From the foreword by DEREK SIVERS

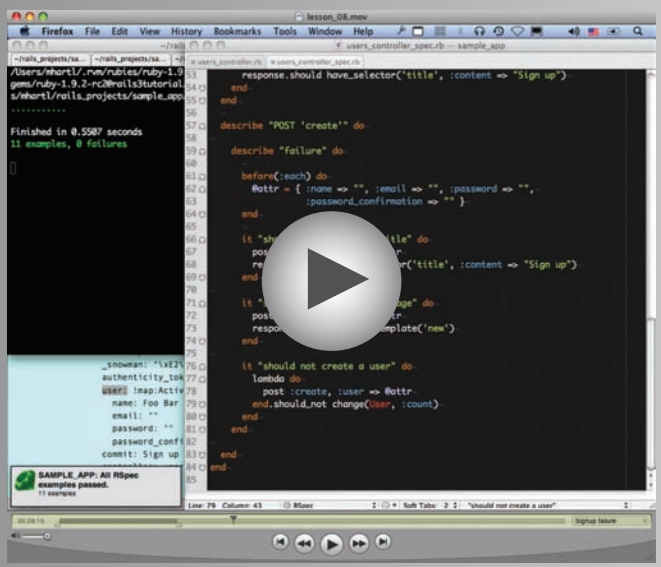
"I got review access to all of the material a week ago and can confirm that, yes, these screencasts are awesome… If you basically want to be able to look 'over the shoulder' of an experienced Rails developer and see how a Rails development environment is set up and how multiple apps are built, there's nothing that can beat this. This isn't a set of 'build a blog in 15 minutes' videos—it's a complete course that could kick off a new career for you with Rails 3.0."

—PETER COOPER, Ruby Inside

GET A FREE LESSON, see testimonials, and read the book in PDF and HTML formats at http://RailsTutorial.org

RECEIVE 10% OFF
any combination of products
ENTER COUPON CODE
"hackermonthly"
valid now through 12.31.10

BUILT FROM THE GROUND UP FOR RUBY ON RAILS 3



- Learn to make real, industrial-strength web applications
- 12 individual lessons totaling more than 15 hours
- Dovetails with, but goes beyond the Rails Tutorial book

Rails Tutorial
railstutorial.org

New Trends In Startup Financing Explained For Laymen

By PATRICK MCKENZIE

Noted American technology investor and all-around smart guy Paul Graham wrote recently about emerging trends in startup funding, specifically that convertible notes and rolling closes are displacing the traditional equity rounds done at a fixed valuation done with angel syndicates.

Did that sound like Greek to you?

Great, you might benefit from this translation of Financier into Geek. (P.S. If you haven't figured out the significance of it originally being written in Financier instead of in Geek, please, think it through.) I originally wrote it as a comment on Hacker News but somebody asked me to put it somewhere easily findable. I have elaborated with standard blog post formating and graphs where I thought they helped the explanation:

Why We Care About Angel Investing

Startups raise money from investors to accelerate their growth into, hopefully, massively profitable businesses and/or massively large acquisitions from big companies.

One particular type of investor that invests in startups is called an angel investor. An angel investor is often an individual human being who is wealthy, frequently as a consequence of successful entrepreneurship. They invest anywhere from $25,000 to $250,000 or so.

Fundraising is painful, and requires a lot of time and focus from startup founders. To mitigate the pain, it is often structured in terms of "rounds", where the startup goes out to raise a particular large sum of money all at once. For an angel round, let's say that could be a million dollars. (n.b. It is trending down, as companies can now be founded for sums of money which would have been laughable a few years ago.) Clearly we're going to need to piece together contributions from a few angels here.

Why Angel Investing Frustrates Founders

Traditionally, one angel has been the "lead" angel, who handles the bulk of the organizational issues for the investors. The rest just sit by their phone and write checks when required. (Slight exaggeration.) Investors are often skittish, and they require social proof to invest in companies, so you often hear them say something like a) they're not willing to invest in you but b) they are willing to invest in you if everybody else does. This leads to deadlocks as a group of investors, who all would invest in the company if they company were able to raise investment, fail to invest in the company because it cannot raise investment.

Startup founders are, understandably, frustrated by this.

What "Valuation" Means

All numbers below this point were chosen for ease of illustration only. They do not represent typical valuations, round sizes, or percentages of companies purchased by angels.

One item of particular interest in investing is the valuation of the company. This gets into heady math, but the core idea is simple: if we agree that the company is worth $100 at this instant in time (the "pre-money valuation"), and you want to invest $100, then right after the company receives your investment, the company is worth $200 (the "post-money valuation"). Since you paid $100, you should own half the company.

Traditionally, the company has exactly one pre-money valuation (which is decided solely by negotiation, and bears little if any relation to what disinterested outside observers could perceive about the company). All investors receive slices in the company awarded in direct proportion to the amount of money they invest. Two investors investing the same amount of money receive the same sized slice of the company. This can be written as "they invested at the same valuation."

The thesis of PG's essay is that allowing investors to invest at the same valuation is not advantageous to the startup. Instead, by offering a discount to valuation for moving quickly, you can convince investors to commit to the deal early, thus starting the stampede from the hesitant investors who were waiting to see social proof.

For example, take the company from earlier. We said it was worth $100 prior to receiving investing, but that is not tied to objective reality. Say instead we'll agree that it is worth $80… but only with respect to the 1st investor. He commits $20. $80 + $20 = $100, so he gets $20 / $100 = 20% of the company for $20, or $1 = 1%. This convinces a second investor to invest. He says "Can I get 20% for $20, too?" Not so fast, buddy, where were you yesterday? The company isn't worth $80 any more. We think it is worth $105 now. (Did we just get through saying $100? Yes. But valuations are not connected to objective reality.) So you get $20 / ($105 + $20) = 16% of the company for your $20. Think that is fair? You do? OK, done.

This continues a few times. The startup raises money—possibly more money, depending on how much the angels want in—with less hassle for the founders.

What Is A Convertible Note? Why Do Founders Like Them?

We've been talking about just dollars so far, and alluding to control of the company as if it were equity like stocks, but there is a mechanism called "convertible notes" at play here. A convertible note is the result of a torrid affair between a loan and an equity instrument. It looks a bit like Mom and a bit like Dad. Like a loan, it charges interest: typically something fairly modest like 6 to 8%, much less than a credit card.

The tricky thing about convertible notes is that they convert into partial ownership of the company at a defined event, most typically at the next round of VC funding or at the sale of the company. So, instead of the first investor getting $20 = 20% of the company, he loans the company $20 in exchange for a promise like this: "You owe me $20, with interest. Don't worry about paying me back right now. Instead, next time you raise money or sell the company, we're going to pretend that I'm either investing with the other guy or selling with you. The portion of the company which I buy or sell will be based on complicated magic to protect both your interests and my interests. If you want to sweeten the deal for me, sweeten the magic."

Do we understand why this arrangement works for both parties? It incentivizes investors to commit early, which lets startups raise more money with less pain. Because startups are in the driver's seat, it also lets them avoid collusion among investors ("We decided we'd all invest in you, but we don't think the company is worth $100. We think it is worth $50. Yeah, that has no basis in objective reality, but objective reality is that your company is worth $0 without the $100 in our collective pockets. What is it going to be? Give up 2/3 of the company, or go broke and get nothing.")

How Do You Calculate The Equity Value of A Convertible Note?

OK, back to complicated magic. When the company takes outside investment, the convertible notes magically convert into stock, based on:

a) the valuation the company receives for the investment round (higher numbers are better for both founders and angels)
b) a negotiated discount to the valuation, to reward the angel investor for his early faith in the company (higher numbers are better for angels)
c) possibly, a valuation cap (higher numbers, or no cap, are better for founders)

For example, continuing with our "low numbers make math comprehensible" startup, let's say it goes on a few months and is then raising a series A round, which basically means "the first time we got money from VCs." We'll say the VC and startup negotiate and agree that the company is worth $500 today, the VC is investing $250, ergo the VC gets a third of the company.

How much does our first $20 angel investor get? Well, he gets to participate like he was investing $20 today, plus he gets a discount to the valuation. So instead of getting $20 / $750 = 2.67% of the company, maybe he got a 20% discount to the valuation, so he gets $20 / (.8 * $750) = 3.33% of the company. (We're ignoring the effect of interest here for simplicity, but he probably effectively has $21 and change invested by now in real life.)

After this is over, the convertible note is gone, and our angel investors are left with just shares (partial ownership of the company), which they probably hold until the company either goes IPO or gets bought by someone. So if the company later gets bought for $2,000 by Google, our intrepid angel investor makes $66 on his $20 investment.

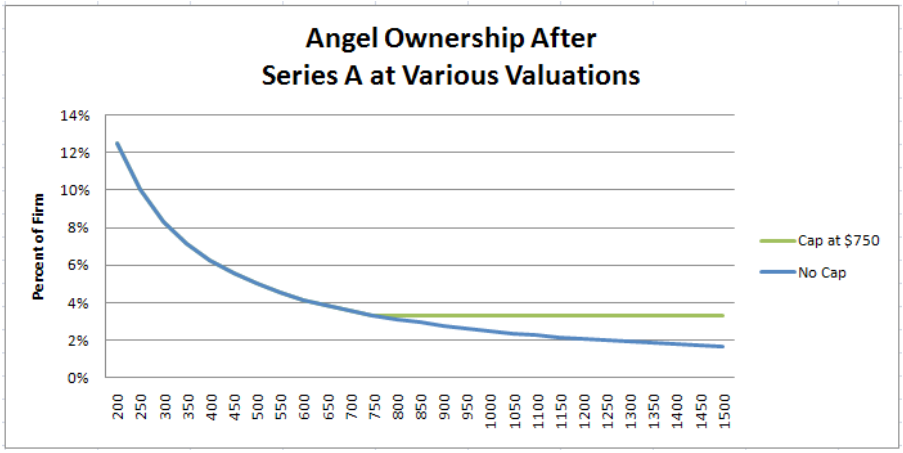
How Does A Valuation Cap Work?

We haven't discussed valuation caps yet. Valuation caps are intended to prevent the startup dragging its feet on raising money, thus building up lots of worth in the company, and then the angel investor getting cheesed. For example, if they had just grown through revenues for a year or two, they might be raising money at a valuation of $1,250. In that case, $20 only buys you 2% of the company (remember, he gets a 20% discount : $20 / (.8 * $1250) = 2%), which the angel investor might think doesn't adequately compensate him for the risk he took on betting on a small, unproven thing several years before. So we make him a deal: he gets to invest his $20 at the same terms as the VCs do if, and only if, the valuation is less than $750. If it is more than $750, for him and only him, we pretend it was $750 instead. This means that under no circumstances will he walk away with less than $20 / (.8 * $750) = 3.33% of the company, as long as the company goes on to raise further investment. (Obviously, if they fold, he walks away with nothing. Well, technically speaking, with debt owed to him by a company which is bankrupt and likely has no assets to speak of, so essentially nothing.)

Perhaps This Will Be Clearer With A Picture

Angels ultimately benefit from higher discounts to the valuation of the Series A round, and lower valuation caps. Higher discounts, and higher effective discounts, mean you get more of the company for less money. That is an unambiguous good, as long as you keep the quality of the company constant.

Let's see how valuation caps affect how much of the company you end up with. The better the company is doing by Series A time, the less of the company the angel ends up with. This shows the incentive for the founders: do as well as you can prior to raising money, which is the same incentive founders always have.



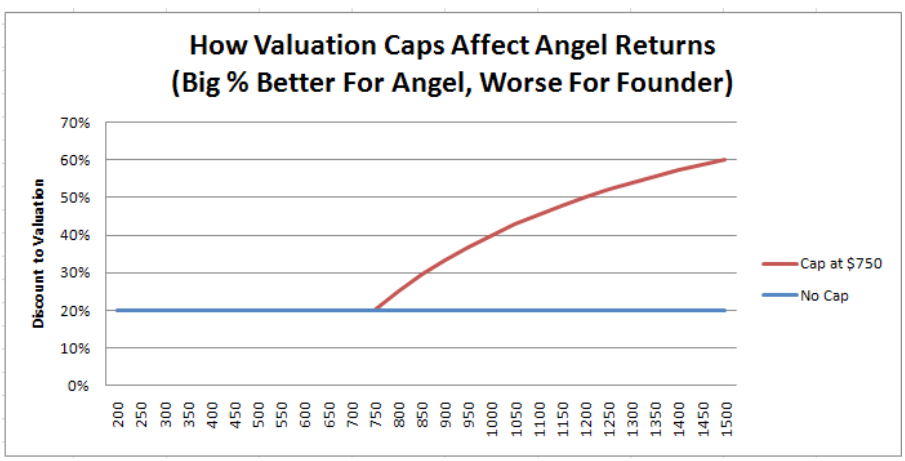
As you can see from the below graph, a valuation cap essentially gives the angel an artificially higher discount for if the Series A valuation exceeds the valuation cap. Obviously then, it is in the interest of angels to negotiate as low a cap as possible, and in the interests of founders to negotiate a high cap or no cap at all. According to Paul Graham, this becomes the primary "pricing" mechanism in the new seed financing economy: if a founder wants to reward an angel, they award them with a lower cap. If they don't, the angels get a higher cap, or no cap at all. This kicks discussions of valuations down the road a little bit, and allows you to simultaneously offer the company to multiple angels at multiple "price points." That allows you to reward them for non-monetary compensation (mentoring, having a big name, etc) or for early action on the deal.

This Is Not My Business. Take With A Grain Of Salt.

Lest anyone get the wrong impression, my familiarity with angel investing is very limited and, to the extent that it exists, it is mostly about angel investing in small town Japan. (Oh, the stories I can't tell.) The above explanation is based on me processing what I've read and trying to prove that I understand it by explaining it to other people. If I have made material errors, please correct me in the comments.

My current business is not seeking funding (and would be an extraordinarily poor candidate for it). I'll never say never for the future, but for the present, I rather like getting 100% of the returns. ■

Patrick McKenzie is a ex-Japanese salaryman who currently runs a small software business. His main product at present is Bingo Card Creator, a product aimed at making elementary school teachers' lives easier.




Duck Duck Go

10 Usability Tips Based on Research Studies

By CAMERON CHAPMAN

We hear plenty of usability tips and techniques from an incalculable number of sources. Many of the ones we take seriously have sound logic, but it's even more validating when we find actual data and reports to back up their theories and conjectures.

This article discusses usability findings of research results such as eye-tracking studies, reports, analytics, and usability surveys pertaining to website usability and improvements. You'll discover that many of these usability tips will be common sense but are further supported with numbers; however, some might surprise you and change your outlook on your current design processes.

1 Forget the "Three-Click Rule"

The idea that users will get frustrated if they have to click more than three times to find a piece of content on your website has been around for ages. In 2001, Jeffrey Zeldman, a recognized authority in the web design industry, wrote that the three-click rule "can help you create sites with intuitive, logical hierarchical structures" in his book, Taking Your Talent to the Web.

Logically, it makes sense. Of course, users will be frustrated if they spend a lot of time clicking around to find what they need.

But why the arbitrary three-click limit? Is there any indication that web users will suddenly give up if it takes them three clicks to get to what the want?

In fact, most users won't give up just because they've hit some magical number. The number of clicks they have to make isn't related to user frustration.

A study conducted by Joshua Porter published on User Interface Engineering found out that users aren't more likely to resign to failure after three clicks versus a higher number such as 12 clicks. "Hardly anybody gave up after three clicks," Porter said.

The focus, then, shouldn't be on reducing the number of clicks to some magically arrived number, but rather on the ease of utility. If you can construct a user interface that's easy and pleasurable to use, but takes like 15 clicks (e.g. 5 times more than the three-click rule) to achieve a particular task — don't let the arbitrary three-click rule stop you.

2 Enable Content Skimming By Using an F-Shaped Pattern

Dr. Jakob Nielsen, a pioneer in the field of usability, conducted an eye tracking study on the reading habits of web users comprising of over 230 participants. What the research study displayed was that participants exhibited an F-shaped pattern when scanning web content.

A similar study, by search marketing firms Enquiro and Did-it in collaboration with eye-tracking research firm Eyetools, witnessed a similar pattern when they evaluated Google's search engine results page with an eye tracking study that included 50 participants. Dubbed the "Google Golden Triangle" because the concentration of eye gazes tended to be top and left, the results are congruent with the F-shaped pattern seen in Nielsen's independent research.

For designers and web copywriters, these results suggest that content you want to be seen should be placed towards the left, and also that the use of content that fits an F-shaped pattern (such as headings followed by paragraphs or bullet points) increases the likelihood that they will be encountered by a user who is skimming a web page.

3 Don't Make Users Wait: Speed Up Your Website

We're always told that our users are impatient: they hate waiting. Well, that's logical — who likes waiting on purpose? But is there any proof outside of anecdotal evidence that people actually don't like waiting and that page performance affects website users?

Bing, Microsoft's search engine, conducted an analysis to see if there are any correlations between page speed and numerical performance indicators such as satisfaction, revenue generated per user, and clicking speed. The report showed that a less than 2-second increase of delays in page responsiveness reduced user satisfaction by -3.8%, lost revenue per user of -4.3% and a reduced clicks by -4.3%, among other findings. For a company as large as Microsoft, even a 4.3% drop in revenue can equate to multi-million-dollar losses in profit.

So users, in fact, are impatient: They're less satisfied and will reduce their number of clicks if they wait too long. And if you care about search engine ranking, then the incentive to improve page response times is even greater since Google now factors page speed in their search ranking.

What can you do to improve page performance? Use tools that will help you find performance bottlenecks, use CSS sprites to improve page speed, and utilize benchmarking tools like YSlow to quickly see where you can make front-end optimizations.

4 Make Your Content Easily Readable

Internet users don't really read content online, at least according to a study by Dr. Nielsen on reading behaviors of people on his website. His analysis shows that people only read 28% of the text on a web page and the percentage decreased the more text there is on the page.

To increase the likelihood of your readers getting the most out of your content, utilize techniques for making content easier to read. Highlight keywords, use headings, write short paragraphs, and utilize lists.

5 Don't Worry About "The Fold" and Vertical Scrolling

There has long been a myth that all of your important content should be above "the fold," a term borrowed from newspapers that refers to the area of a web page that can be seen without having to scroll down — first proposed by Jakob Nielsen.

So, are long pages bad? Should we cram everything at the top of our web layouts because people won't ever read anything below this fold?

The answer is "No" according to a report by Clicktale, a web analytics company. Their results showed that the length of the page has no influence in the likelihood that a user will scroll down the page.

A study reported by Joe Leech of CX Partners, a user centered design agency, indicated that less content above the fold even encourages users to explore the content below the fold.

The main point to take away here is that you shouldn't stuff all your important content at the top because you fear that users won't be able to find them otherwise. Use visual hierarchy principles and the art of distinction to prioritize and infer the importance of various elements in your pages' content.

6 Place Important Content on the Left of a Web Page

People brought up in cultures where language is read and written from left to right have been trained early on in life to begin at the left of a page, whether in writing or reading a book. This can be the reason why many web users spend a majority of their attention on the left side of a web page — as much as 69% of the time, according to Dr. Nielsen's eye-tracking study that involved over 20 users.

The same results were reflected on websites whose language were read from right to left, such as Hebrew and Arabic sites, with the results inverted (higher attention on the right side versus the left).

There are two things to take away from this result. First, the language of your site matters when thinking about layout considerations; when designing websites you should consider cultural design considerations. Secondly, for sites that are traditionally read from left to right, placing important design components at the left is a good idea; vice versa for sites whose language is read from right to left.

7 Whitespace of Text Affects Readability

Easy readability of text improves comprehension and reading speed as well as enhancing the likelihood that a user will continue reading instead of abandoning the web page. There are many factors that influence ease of readability, including font choices (serif versus sans-serif), font-size, line-height, background/foreground contrast, as well as spacing.

A study on readability tested reading performance of 20 participants by presenting them with the same text blocks having different margins surrounding the text as well as varying line-heights (the distance between lines of text). It showed that text with no margins was read faster, however, reading comprehension decreased. Faster reading speeds when the text had no margins can be explained by the text and paragraphs being closer together, resulting in less time needed to move the eyes from line to line and paragraph to paragraph.

As this particular study shows, the way we design our content can greatly impact the user's experience. Be wary of the details: color, line-heights, tracking, and so forth and be mindful of sound typography principles for the web to ensure that you're not discouraging your users from reading your content. Furthermore, study the effective use of negative space in web design.

8 Small Details Make a Huge Difference

Too often, we look at the big picture when creating a web design and ignore the little things when we're in a time crunch. We forego any thought put into the wording of something, or the design of a single button on a form if time and resources are limited. There are so many other things we need to think about that it's often easy to let go of the small stuff.

But something as small as a form's button can affect the success of a site, at least according to user interface design expert Jared Spool, who wrote about a case where removing a button and adding a clear error message to avoid user errors in a checkout process increased revenue by $300 million in just a year. After the revision of the checkout process, customers purchasing went up by 45%, generating $15 million in the first month.

Flow, a user-centered design firm, echoes Spool's emphasis on the importance of attention to detail. They found that revising an error page so that it contained useful help text improved completed checkouts by 0.5% per month, which if extrapolated, could mean an additional quarter of a million pounds annually for the particular site.

The message they used? A polite two-sentence message instead of a cryptic 404 error: "We're sorry, we've had a problem processing your order. Your card hasn't been charged yet. Please click checkout to try again."

Pay attention to the details. Use A/B split testing to test your hypothesis and find out what is the most effective design that achieves better results. Set goals using analytics software to benchmark results of design tweaks in relation to site objectives.

9 Don't Rely on Search as a Crutch to Bad Navigation

Users expect navigation to be easy to use and well organized. Even with an excellent site search engine, users will still turn to primary navigation first. According to a task test conducted by Gerry McGovern, over 70% of the participants began the task he gave them by clicking on a link on the page as opposed to using the search feature.

This result is similar to a test by UIE of 30 users that tracked e-commerce tasks. The research analysis concluded that "users often gravitated to the search engine when the links on the page didn't satisfy them in some way." Thus, search is most often utilized only when the user has failed to discover what they were looking for in the current page.

The lesson to be gained here is clear: Don't rely on site search to remedy poor content organization, findability issues, and bad information architecture. When users are unable to navigate to what they are looking for, attention should be diverted to layout, navigation, and content organization improvements, with improving search functionality as the secondary priority.

10 Your Home Page Isn't As Important as You Think

Visitors to your website are less likely to land on your home page. Search engines are a big factor here, as they'll link to whatever page is relevant on your site. Links from other websites are also likely to link to pages beyond your home page if that's where the relevant information is.

According to an analysis by Gerry McGovern, page views sourcing from the home page of websites is decreasing dramatically. He witnessed a drop from 39% from 2003 to only 2% in 2010 of page views coming from the home page of a large research site. This trend was doubly confirmed on another site he studied, where page views sourcing from the home page halved in just two years (from 10% in 2008 to only 5% in 2010).

McGovern's results indicate that traffic, more and more, is coming from external sources — search engines, social media sites such as Twitter, and content aggregator services such as AllTop — rather than from the front page of a website. Therefore, a higher focus on landing pages versus your home page can get you more bang for your buck in terms of conversion and user-retention opportunities. ■

Cameron Chapman is a professional web and graphic designer with over 6 years of experience in the industry. She's also written for numerous blogs such as Smashing Magazine and Mashable. You can find her personal web presence at Cameron Chapman On Writing [http://cameronchapman.com/]. If you'd like to connect with her, check her out on Twitter @ cameron_chapman.

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

The Accidental Launch

By RAHUL VOHRA

We accidentally got 10,000+ users in 24 hours, and funding from Y Combinator just a few days later. This post tells that story.

We were determined to take part in Y Combinator, so we spent weeks crafting our entry and polishing Rapportive. At the start of March, we were finally ready. We held our breath and clicked "Submit." We looked at each other, relaxed, and slowly started to breathe again. A few hours passed uneventfully. We were in no way prepared for what happened next.

Somehow, the press had found us. TheNextWeb ran the first piece. ReadWriteWeb picked it up after that. Then Lifehacker. Then WebWorkerDaily. We had headlines like: "Stop What You Are Doing & Install This Plug-In." Our twitter account was aflame with thousands of mentions in just a few hours. We had accidentally launched.

We saw our user count grow from 5 to over 10,000 in 24 hours. I had a case of beers in my drawer in case we ever needed to celebrate anything. We drank all of them.

I stayed awake for two days straight: the emails didn't slow down, the tweets kept pouring in, and new Skype chats would appear as soon as I'd finish old ones. But we were determined to quickly respond to every single last email, tweet, and chat, so we soldiered on.

The next day, investors from across the world started contacting us with offers of funding. These weren't just any old investors; these were some of the best angels and venture capitalists in the world.

We didn't have time to wait for the normal Y Combinator interview, which would have happened a month later. I contacted Harj, Venture Partner at YC, and they offered to do the interview over Skype. (I vaguely knew Harj from our university days — it's a surprisingly small world.)

A few days later, Martin, Sam and I were huddled around around a laptop talking to pg, Jessica and Harj. They weren't quite as huddled, so we spent most of it talking to pg's legs. We talked for half an hour, but I felt like it passed by in an instant. A few minutes later, we had our answer: Y Combinator would fund us!

We celebrated in the traditional British manner. When we were next coherent, we booked a fundraising trip to the Valley.

Lessons Learnt

We did several things that worked well during this phase:

- Offer surprisingly great service. Most companies deliver terrible service, and users have come to expect it. Surprise them. Make it abundantly clear how users can contact you. Monitor all your channels. Respond to people as soon as you physically can. Thank everybody and go the extra mile. I personally find that it really helps to smile, even when the user is thousands of miles away and on the other end of a tweet. We use a shared Gmail account for email support, and CoTweet for twitter. Our YC batchmates rave about Olark.
- Use a feedback forum. Make the forum really easy to find. Include links to it from your product. Make the links especially visible when the product isn't working properly. If your forum provides single sign-on (so users don't have to create new accounts) then use it! We use UserVoice and have fallen irrevocably in love with it.
- Release early. We didn't choose to release early: it was a complete accident! But in hindsight it turned

out to be very useful. Our feedback forum rapidly filled up. We quickly learnt peoples' likes and dislikes, and prioritised building what people want. If you don't release early, then you might build the wrong thing and you won't find out until much later. Even if you build the right thing, somebody else might build it first and steal your thunder. So get out there.

- Be ready to scale. You never know when traffic will hit. Now I realise that "be ready to scale" may sound like classically bad advice, but cloud computing has changed the economics. You can be ready by simply choosing the right hosting provider. If we were on a cheap VPS, we would have crumbled to pieces like Cobb's limbo in Inception. As we were on Heroku, we could simply increase the number of dynos. I still vividly remember when our traffic hit. I was away from my desk, so I reached for my iPhone and dialed us up to 20 dynos using Nezumi. A few seconds later, we had scaled.
- Build for the press. It turns out that Rapportive works exceedingly well for technology bloggers, because they spend so much time corresponding with people who have significant online presences. It is not worth building functionality only for the press (unless, of course, they are your target market), but it is worth being aware of this effect.
- Build early. This advice is specifically for companies applying to Y Combinator: start as early as you can, as the deadline will come soon. The most impressive thing you can do is make something that people want.

One of our favourite books is Founders at Work, a collection of interviews with founders about their early days. We're now collecting stories of our own, which we will post in a series, Rapportites at Work. This post is the first of the series. ■

Rahul Vohra is a co-founder and the CEO of Rapportive. He's a computer scientist, a gamer, and an entrepreneur. You can follow Rahul on twitter at http://twitter.com/rahulvohra

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

Most Common Words Unique to 1-star and 5-star App Store Reviews

By MARCO ARMENT

I WROTE A SCRIPT to crawl U.S. App Store customer reviews for the top 100 apps from every category (minus duplicates) and compute the most common words in 1-star and 5-star reviews, excluding words that were also common in 3-star reviews.

Keep in mind that the results are not representative of overall user opinions: most users don't review apps, and people who dislike an app are more likely to leave a review than people who like it.

These are the top words by rating, with descending frequency:

★★★★★:
awesome, **worth**, thanks, **amazing**, **simple**, **perfect**, price, everything, ever, must, ipod, before, found, store, never, recommend, done, take, always, touch

★☆☆☆☆:
waste, money, **crashes**, tried, **useless**, nothing, paid, open, deleted, downloaded, didn't, says, **stupid**, anything, actually, account, bought, apple, already

Bold words are adjectives or likely to be used as adjectives in context.

Some are obvious: people like awesome apps and dislike those that crash. A few words are more interesting, though:

It's promising to see **simple** in the top-positive list, which says a lot about user expectations on the platform.

Both positive and negative reviews seem unusually obsessed with price. This seems odd, given the relative cost of the hardware, accessories, and cellular service where applicable.

The negative words are most interesting to me: in addition to complaints about the price, one word is especially telling of a prevalent attitude I've seen for a while: **useless**. More than any other adjective, reviewers condemn apps they don't like as "useless." Subjectively, I usually see this in contexts in which the app doesn't have a minor feature that the reviewer wants, or where it doesn't perform well in a rare use-case, so the reviewer unfairly declares the app "useless." This demonstrates a curious psychological effect of modern western culture that I'll write about soon. ■

Marco Arment is the founder of Instapaper and the former cofounder of Tumblr. He converts coffee and Phish to web and iOS apps, and he writes at Marco.org.

Reprinted with permission of the original author. First appeared in *http://hn.my/appstorewords/*.
Icon by Ricky de Laveaga.

Regrets of the Dying

By BRONNIE WARE

For many years I worked in palliative care. My patients were those who had gone home to die. Some incredibly special times were shared. I was with them for the last three to twelve weeks of their lives.

People grow a lot when they are faced with their own mortality. I learnt never to underestimate someone's capacity for growth. Some changes were phenomenal. Each experienced a variety of emotions, as expected, denial, fear, anger, remorse, more denial and eventually acceptance. Every single patient found their peace before they departed though, every one of them.

When questioned about any regrets they had or anything they would do differently, common themes surfaced again and again. Here are the most common five:

1 I wish I'd had the courage to live a life true to myself, not the life others expected of me.

This was the most common regret of all. When people realise that their life is almost over and look back clearly on it, it is easy to see how many dreams have gone unfulfilled. Most people had not honoured even a half of their dreams and had to die knowing that it was due to choices they had made, or not made.

> **“When you are on your deathbed, what others think of you is a long way from your mind.”**

It is very important to try and honour at least some of your dreams along the way. From the moment that you lose your health, it is too late. Health brings a freedom very few realise, until they no longer have it.

2 I wish I didn't work so hard.

This came from every male patient that I nursed. They missed their children's youth and their partner's companionship. Women also spoke of this regret. But as most were from an older generation, many of the female patients had not been breadwinners. All of the men I nursed deeply regretted spending so much of their lives on the treadmill of a work existence.

By simplifying your lifestyle and making conscious choices along the way, it is possible to not need the income that you think you do. And by creating more space in your life, you become happier and more open to new opportunities, ones more suited to your new lifestyle.

3 I wish I'd had the courage to express my feelings.

Many people suppressed their feelings in order to keep peace with others. As a result, they settled for a mediocre existence and never became who they were truly capable of becoming. Many developed illnesses relating to the bitterness and resentment they carried as a result.

We cannot control the reactions of others. However, although people may initially react when you change the way you are by speaking honestly, in the end it raises the relationship to a whole new and healthier level. Either that or it releases the unhealthy relationship from your life. Either way, you win.

4 I wish I had stayed in touch with my friends.

Often they would not truly realise the full benefits of old friends until their dying weeks and it was not always possible to track them down. Many had become so caught up in their own lives that they had let golden friendships slip by over the years. There were many deep regrets about not giving friendships the time and effort that they deserved. Everyone misses their friends when they are dying.

It is common for anyone in a busy lifestyle to let friendships slip. But when you are faced with your approaching death, the physical details of life fall away. People do want to get their financial affairs in order if possible. But it is not money or status that holds the true importance for them. They want to get things in order more for the benefit of those they love. Usually though, they are too ill and weary to ever manage this task. It is all comes down to love and relationships in the end. That is all that remains in the final weeks, love and relationships.

5 I wish that I had let myself be happier.

This is a surprisingly common one. Many did not realise until the end that happiness is a choice. They had stayed stuck in old patterns and habits. The so-called 'comfort' of familiarity overflowed into their emotions, as well as their physical lives. Fear of change had them pretending to others, and to themselves, that they were content. When deep within, they longed to laugh properly and have silliness in their life again.

When you are on your deathbed, what others think of you is a long way from your mind. How wonderful to be able to let go and smile again, long before you are dying.

Life is a choice. It is YOUR life. Choose consciously, choose wisely, choose honestly. Choose happiness. ■

Bronnie Ware is a writer and singer/songwriter from Australia. She is currently writing a book on her experiences in palliative care and is working on a new album of inspirational songs.

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

The Most Powerful Colors in the World

By DARIUS A MONSEF IV

When we released our report on the colors of the social web, based on data analyzed by our Twitter theme tool, we were surprised that blue was such a dominant color in people's profile designs. Was Twitter's default color influencing their design decisions? Or is blue really THE most popular and dominant color online? ...We decided to look at the colors in the brands from the top 100 sites in the world to see if we could paint a more colorful picture.

Turns out the blueberry doesn't fall far from the bush. The web landscape is dominated by a large number of blue brands... but Red occupies a large amount of space as well. What's driving this? You might want to say that carefully organized branding research and market tests were done to choose the perfect colors to make you spend your money, but a lot of the brands that have grown to be global web powerhouses, started as small web startups... and while large corporate giants with branding departments spend quite a lot on market research, user testing, branding, etc., lots of the sites listed above got started with brands created by the founders themselves with little to no research into the impact their color choice would have. I once asked Mark Zuckerberg, the founder of Facebook why he chose blue for his site design... "I'm color blind, it's the only color I can see." ...and now 500 Million people around the world stare at a mostly blue website for hours each week.

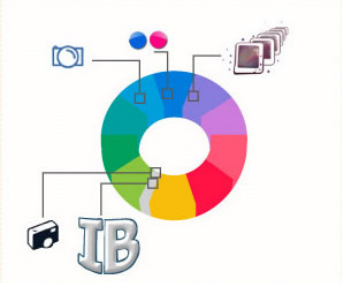
While the initial reasoning for the colors chosen may be trivial, the impact that these dominant players now have in the web world will surely influence the smaller startups that want to share in the positive color associations created by their bigger siblings... Once a rocketship of a web startup takes flight, there are a number of Jr. internet astronauts hoping to emulate their success... and are inspired by their brands. And so Blue and Red will probably continue to dominate, but we can have hope for the Gowalla's, DailyBooth's and other more adventurous brands out there.

TOP WEB BRAND RANKINGS:

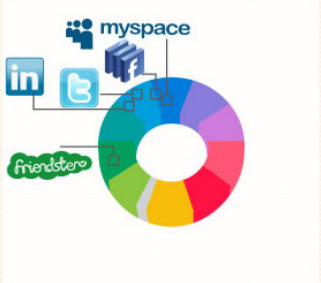
1 GOOGLE SEARCH PORTAL	21 WEATHER CHANNEL WEATHER	41 ORKUT SOCIAL NETWORK	61 LIMEWIRE FILE SHARING	81 PRICELINE TRAVEL
2 MSN/WINDOWSLIVE/BING SEARCH PORTAL	22 GLAM MEDIA MEDIA/NEWS	42 CHASE FINANCIAL	62 WEBMD HEALTH	82 EXPERIAN FINANCIAL
3 YAHOO! SEARCH PORTAL	23 CNN MEDIA/NEWS	43 UOL PORTAL	63 FRIEND FINDER ONLINE DATING	83 PORNHUB ADULT
4 MICROSOFT SOFTWARE	24 TWITTER SOCIAL NETWORK	44 BANK OF AMERICA FINANCIAL	64 SHOPPING RETAIL	84 IVILLAGE PORTAL
5 FACEBOOK SOCIAL NETWORK	25 SKYPE VOIP	45 EHOW REFERENCE	65 NIKELODEON CHILDREN	85 UPS RETAIL
6 YOUTUBE VIDEO	26 CBS MEDIA/NEWS	46 LIVEJASMIN ADULT	66 CLASSMATE SOCIAL NETWORK	86 SUPERPAGES REFERENCE
7 WIKIPEDIA REFERENCE	27 IMBD REFERENCE	47 ESPN MEDIA/NEWS	67 NETFLIX VIDEO	87 FOX NEWS MEDIA/NEWS
8 AOL SEARCH PORTAL	28 WALMART RETAIL	48 ZYNGA GAMING	68 MEEBO SOCIAL NETWORK	88 NFL SPORTS
9 EBAY RETAIL	29 CRAIGSLIST RETAIL	49 SHOPZILLA RETAIL	69 SIX APART BLOGGING	89 DAILYMOTION VIDEO
10 APPLE RETAIL	30 BBC MEDIA/NEWS	50 COMCAST MEDIA/NEWS	70 TURNER SPORTS MEDIA/NEWS	90 T-ONLINE PORTAL
11 AMAZON RETAIL	31 TERRA PORTAL	51 VIDEOLAN SOFTWARE	71 TUDOU VIDEO	91 REED BUSINESS NEWS/MEDIA
12 BLOGGER BLOGGING	32 CNET TECH NEWS	52 EVERDAY HEALTH HEALTH	72 HEWLETT PACKARD COMPUTER	92 PIRATE BAY FILE SHARING
13 ASK SEARCH PORTAL	33 ORANGE PORTAL	53 LINKEDIN SOCIAL NETWORK	73 NEXTAG RETAIL	93 CITIBANK FINANCIAL
14 FOX INTERACTIVE MEDIA MEDIA/NEWS	34 DISNEY ONLINE PORTAL	54 EXPEDIA TRAVEL	74 NBC MEDIA/NEWS	94 VISTAPRINT COMMERCE
15 MOZILLA SOFTWARE	35 AT&T PORTAL	55 IG FINANCIAL	75 CONDUIT SOFTWARE	95 SEARS RETAIL
16 REAL NETWORK SOFTWARE	36 NETSHELTER TECH TECH NEWS	56 TARGET RETAIL	76 VERIZON PORTAL	96 TRIBUNE NEWSPAPERS MEDIA/NEWS
17 ADOBE SOFTWARE	37 FLICKR PHOTO SHARING	57 DELL RETAIL	77 TRIP ADVISOR TRAVEL	97 EA GAMES GAMING
18 ABOUT REFERENCE	38 PICASA PHOTO SHARING	58 GLOBO MEDIA/NEWS	78 BEST BUY RETAIL	98 MEGAUPLOAD FILE STORAGE
19 PAYPAL FINANCIAL	39 GORILLA NATION COMMERCE	59 SCRIPPS NETWORK MEDIA/NEWS	79 MONSTER EMPLOYMENT	99 VODAFONE PORTAL
20 WORDPRESS BLOGGING	40 WIKIANSWERS REFERENCE	60 NYTIMES MEDIA/NEWS	80 RTL NETWORK MEDIA/NEWS	100 GEEKNET FORUM

TOP BRANDS WITHIN CATEGORIES TEND TO USE SIMILAR COLOR PALETTES:

PHOTOS



SOCIAL NETWORKS



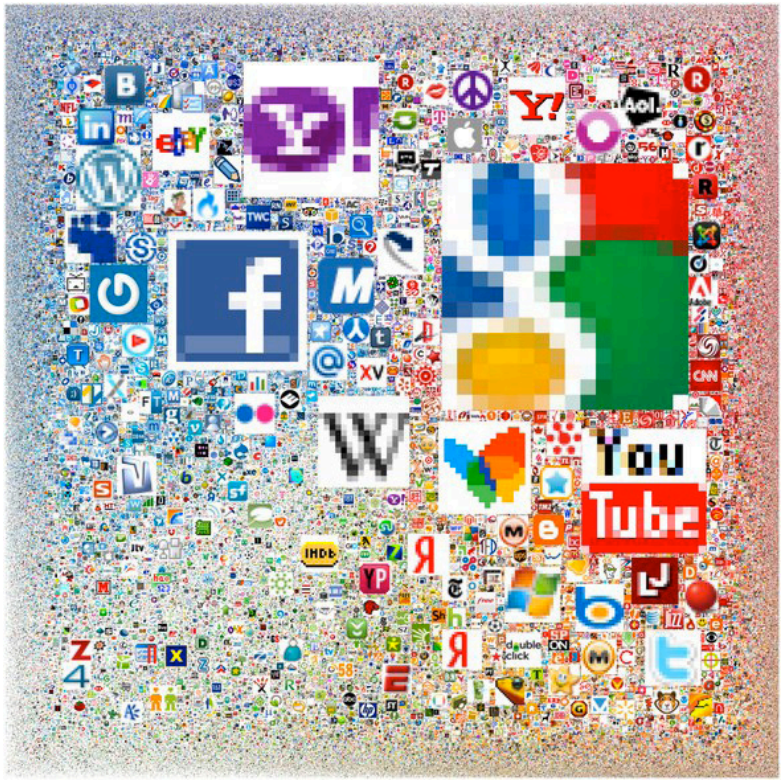
BLOGGING

GEOTAGGING

SOCIAL NEWS

SOURCES: ALEXA | COMPETE | NIELSON

http://www.colourlovers.com/business/blog/2010/09/15/the-most-powerful-colors-in-the-world



Would A Corporation By Any Other Color, Still Profit As Well?

Color is an important part of any brand, but along with the actual name of a company... Is it a great brand that builds a great company, or the other way around? Would Google, Google just as well with another name? My guess is yes.

And almost 10 years ago, Wired Magazine looked at the Colors of the corporate America... Blue & Red dominate again.

> *Companies spend millions trying to differentiate from others. Yet a quick look at the logos of major corporations reveals that in color as in real estate, it's all about location, location, location. The result is an ever more frantic competition for the best neighborhood. Here's a look at the new blue bloods.*
> *[Wired Magazine]*

The Colors of 1 Million Brand Icons

And a brand can extend further than just your logo... On the web it reaches into the address bar in the form of a Favicon. It's quite amazing to explore, but the top 1,000,000 website Favicons can be browsed here at Icons of the Web [http://nmap.org/favicon/].

Uh-oh! But Will We Run Out of Color on the Web?

Last year Francisco Inchauste posted a very interesting article on SixRevisions about the limited resource of color... not in physical form, but in mind share. (Even linking to a post we did a while back about T-Mobile and it's trademark of "Magenta")

> *As a designer, it is important to be aware of the trending colors, and how they are being applied in products and work produced today. What really isn't being discussed by the design world at large though are the limitations being set on color. Color is as free for us to use as the air we breathe… or is it? [SixRevisions]*

The Next Big Color Trend

You are the next great founder, designer, influencer or creative mind that may build the next Facebook. You have the power to influence future color trends... What colors will you choose? ■

Darius wants the whole world to find color enlightenment. He is the CEO of CHROMAom, Inc and the creator of COLOURlovers.com. He built COLOURlovers after an uninspired class on color theory left him searching for an online community to explore color. Darius is an internet entrepreneur, web designer/coder, former student of fashion design and was previously community organizer for Microsoft's Photosynth software. He's also the co-founder of the disaster relief non-profit Hands On Disaster Response (HODR) and has spent more than 11 months living in disaster zones around the world.

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

How Do I Write So Much

By SEBASTIAN MARSHALL

A few of my friends – three friends, to be exact – mentioned to me that I write a heck of a lot on here and they're impressed. I have convinced the ultra-smart Sami Baqai to start blogging, and he just got the holy-shit-this-is-hard-I'm-overwhelmed feeling. Ah, yes, I have been there Sami. Perhaps I can share some thoughts.

First and foremost, I am a huge devotee of the Equal-Odds Rule. As far as I know, I'm the only person talking about it outside of academia. This Amazon review covers it pretty well:

> *The equal-odds rule says that the average publication of any particular scientist does not have any statistically different chance of having more of an impact than any other scientist's average publication. In other words, those scientists who create publications with the most impact, also create publications with the least impact, and when great publications that make a huge impact are created, it is just a result of "trying" enough times. This is an indication that chance plays a larger role in scientific creativity than previously theorized.*

So I read that, and I'm like – whoa. You know Neo in the Matrix? Whoa.

If you want to make excellent stuff, you need to make a lot of stuff.

If you want to make a lot of stuff, you'll make a lot of crap.

If you want to make excellent stuff, you need to make a lot of crap.

And my personal opinion here —

And that's okay, because you get judged by your best work, not your bad work.

At the risk of being honest, a lot of my writing here is crap. I mean, it's okay, it's not totally stupid, but a lot of it is very "meh" – well, by own estimation. But occasionally I really nail something, and that's what people are going to remember. A Lot of Victory is Just Walking Around turned out to be a huge hit and got hundreds of visitors from people Facebook-liking it, when I just typed it up on the spur of the moment. I thought it was good, but nothing crazy revolutionary – I was talking about noticing where business are in certain areas, and what businesses are missing that you could potentially build. I talked about putting a premium mechanic shop in an upscale district of Hong Kong I was walking around, or opening a coffee chain in Cambodia. People loved that, I got so many compliments and lots of new visitors, many of whom stuck around and are still readers. (Hi guys! Glad you stuck around) In retrospect, I guess yeah that was a good post. But it only happened because I wrote some very just-okay posts too.

Alright, but let's talk nuts and bolts more. Three things we've already covered this post —

1. I believe in the Equal-Odds Rule, which states roughly that a creator can't entirely control the quality of their output. In order to do high impact excellent work, you have to do a lot of work, which includes low impact not excellent work.

2. I think as long as you're not doing life-or-death stuff, it's okay to put out low quality work. Well, not really. I'm kind of a perfectionist. What I actually mean is you're going to be a bad judge of how good your own stuff is, especially if it's creative work. Don't put out anything wrong or terrible or lazy, but if something is okay and you gave it your best, put it out. People might like it, or might not, but you probably won't be able to know in advance.

3. You'll get judged by your best work. I've written up at least 150 articles over the last four months. If I want to present my writing to someone, I'll link to the best 10-20 and get evaluated on those. If I'm pitching something really important, I can always go edit and polish an even better version.

“You’re going to have to put some crap out to do great work.”

This is big stuff. This is the mental side of it. I happen to know how good Sami’s writing is, because he and I swap emails and share ideas. We connected originally from Hacker News, and he’s a super-sharp guy, very multi-disciplinary bright. But Sami obviously got some issues putting crap out into the world. He doesn’t want do it. Well, Sami, you want to do great work or not? You’re going to have to put some crap out to do great work. I know, it’s hard. It sucks. Mind you, I don’t want to put crap out. It’s just, that’s the Equal-Odds Rule, which I am a believer in.

Alright, nuts and bolts for real this time.

4. I commit to doing it every day, every single day no matter what.

5. My audience is whoever likes it – the site is written for me. If someone doesn’t like it at this point in their life, they’re not my audience for now.

6. Extensive notes/backlog – quotes, stories, pictures, ideas. Lots of this.

7. I accepted that I’m going to judged. I don’t love it, but I accepted it. It comes with the territory.

8. Look at my first entries if you want to be inspired. Or any blogger’s first entries. Or Seth Godin’s “E-Marketing” book from 1995. Sort of cheesy – “MORE THAN $1,000 WORTH OF MONEY-SAVING COUPONS INSIDE” – but it doesn’t seem to have derailed his career. Just the opposite, actually – we all gotta start somewhere.

   A few tactical thoughts:

9. Post scheduling is good, especially if not going to be near internet. You can schedule when a post comes live pretty easily on any modern blogging platform. I don’t like to do this too far in advance, because I want my currently published things to be whatever I want to talk about on the phone with people or in email since people do bring it up. But I often write a post before sleeping, and schedule it to go live a minute after midnight. That way, I’m not under time crush the next day to make sure I get a blog post in. If I want to write more, I’ll write a second post that day. If I’m not sure about internet because I’m flying, I’ll schedule two in a row, one for the next day, one for the day after, but I don’t even do that too often. I like my writing to be whatever is on my mind.

10. Not worrying about perfection, just starting.

11. Try to think of every visitor as an honored guest. If you think of “web traffic,” 15 visitors is disappointing. If you think of 15 people deciding to spend time with you they could spend anywhere, and they’re choosing to spend it with you – they’re choosing to spend their life energy reading your thoughts – that’s very cool and humbling, and suddenly chugging along with 15 readers feels pretty good. I had between 10 and 40 visitors for the longest time. The site is starting to blow up a little bit more, had 746 unique visitors on September 1st and have been above 200 daily visitors consistently recently, but I was pretty honored even when 10 people were stopping by for 4 minutes each. That’s 40 minutes of life energy people are choosing to spend with you instead of somewhere else. Like, that’s pretty humbling. Now I have 200 regular readers? Like, whoa. That’s 800 minutes per day. People are spending 12 hours of life-time each day with me. Wow. That’s cool. Even when it was 10 per day, I was thinking that was really cool and humbling.

12. On a very busy day, I’ll just post a quote or a short insightful thought. I’ve got some quotes from Miyamoto Musashi, Tokugawa Ieyasu, Marcus Aurellius, Thomas Jefferson, Sun Tzu, Carl von Clauswitz, and others lined up.

    “The strong manly ones in life are those who understand the meaning of the word patience. Patience means restraining one's inclinations. There are seven emotions: joy, anger, anxiety, adoration, grief, fear, and hate, and if a man does not give way to these he can be called patient. I am not as strong as I might be, but I have long known and practiced patience. And if my descendants wish to be as I am, they must study patience.” – Tokugawa Ieyasu

13. Listen to audio at cafes with nothing else to do. Sit there, have coffee, listen to smart audio. Ideas will come. Jot down a note.

14. When you have a good idea, write it down. I have a “shorttermblog.txt” on the desktop of my laptop, and there’s at least dozens of ideas written down in there. Sometime or other I’ll talk about Roman Emperor Septimus Severus made a huge mistake making his two sons Caracella and Geta joint-Emperors. Dude, Septimus, that never works…

15. Have fun. I mean, really have fun. Look at my “Some General Life Goals” – “carrying self like rich dickhead” is on the list. After I already took a screenshot of my computer, I realized that was

> **“Are you such a big deal that you can’t be embarrassed, or make a mistake, or do something wrong?”**

on there. I thought about censoring it. Nahh, whatever. Someone could judge me? Yes. Someone could get offended? Yes. I just wrote up another post, “Arguing With Peasants Shows a Lack of Self-Discipline” – I thought to myself, “Do I really want to write that?” Am I going to get asked on some news interview sometime, “So, you think you shouldn’t argue with peasants, do you?” in a really sanctimonious, judging tone that makes me look bad? I don’t know, maybe. Probably? Whatever. It’s actually how I think. I read some insight from economist Vilfredo Pareto about how the peasants never actually take control of the government, instead one elite uses the peasants to kill off the other elite, but the peasants themselves never take power. Reading that, a lot of things clicked. I said, “Ohhh, I shouldn’t argue with peasants who believe they can really take power.” A lot of peasants are backing their team – well, have fun in your new worker’s paradise Socialist Soviet Republic. Idiots. Will I catch flak later because I shared my honest opinion about this? Maybe. But whatever, it’s how I think. This is a relatively new feeling for me, in the past I always tried to be diplomatic, and now I’m more and more just saying what I’m actually thinking. It’s actually enjoyable in its own strange way.

16. That leads me to the final point, which is you gotta remember this is all a circus. Life is really a circus. Are you such a big deal that you can’t be embarrassed, or make a mistake, or do something wrong? No, you’re not. You’re not a big deal. At least, I’m not a big deal. I’ll say some stupid shit at some point, and get embarrassed, and look bad. Oh well. If things break the right way, I’ll also found branches of science, inspire people, build amazing businesses, found charities that actually work, make art, fund art, fund science, build a virtuous international dynasty, and all sorts of other stuff. But if I try and fail? Well, whatever, I’m not such a big deal. I can be embarrassed. It’s okay if I get something wrong or say something stupid. Most of what we obsess over is going to turn to dust anyways.

    My favorite poem: Ozymandius by Percy Bysshe Shelley

    I met a traveller from an antique land
    Who said: Two vast and trunkless legs of stone
    Stand in the desert. Near them, on the sand,
    Half sunk, a shattered visage lies, whose frown
    And wrinkled lip, and sneer of cold command
    Tell that its sculptor well those passions read
    Which yet survive, stamped on these lifeless things,
    The hand that mocked them and the heart that fed.
    And on the pedestal these words appear:
    “My name is Ozymandias, king of kings:
    Look on my works, ye Mighty, and despair!”
    Nothing beside remains. Round the decay
    Of that colossal wreck, boundless and bare
    The lone and level sands stretch far away.

    This is all coming down, man. Turning to dust. Life’s a circus.

    Now, some people have this attitude of, “Well, all this doesn’t matter, so I’m just going to party, or do nothing, or whatever.” Me? No way! I think, “Well, most of this doesn’t matter, so I might as well found branches of science, do great works, build amazing things, make art, write, fund things, build things, fix things, serve people, and otherwise do amazing stuff.”

    I mean, why not, right?

    On the tactical level, I’d strongly recommend committing to writing every day. Every single day, something. Even something small. People liked “Sun Tzu says – Make It Look Easy” and that was just a short quote I picked up listening to the Art of War.

    Look at my early posts, if you like. A lot of them aren’t very good. But you start doing it every day, every single day, and you get better pretty quickly. You start noticing what people like, and tweaking your works, and it’ll come. Just accept that your early work is going to suck, and even later some of your work is going to suck, and cherish every visitor. I’ll add you to my RSS reader and I’ll stop by from time to time, so there, you’ve got at least one visitor. Do it every day, eh? You’ll suck and make crap for a while, and then you’ll do good stuff, and in not-very-long you’ll do some awesome stuff. Tone is hard to get, but it comes with time. Every single day is the way. Something, even just a quote. You’ll find the theme later. Now, get started, eh? ■

Sebastian Marshall has stated his goal is to train to be the greatest strategist of this generation. He writes on strategy, entrepreneurship, technology, business, marketing, philosophy, history, governance, and creativity at SebastianMarshall.com, a site that is updated daily with new insights. He prides himself on being very accessible and helpful – feel free to shoot him a line with a question, comment, or feedback at sebastian@sebastianmarshall.com.

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

The Treacherous Optimization

Old age and treachery will beat youth and skill every time.

By RIDICULOUS_FISH

"I'M GOING TO beat grep by thirty percent!" I confidently crow to anyone who would listen, those foolish enough to enter my office. And my girlfriend too, who's contractually obligated to pay attention to everything I say.

See, I was working on Hex Fiend, and searching was dog slow. But Hex Fiend is supposed to be fast, and I want blazingly quick search that leaves the bewildered competition coughing in trails of dust. And, as everyone knows, the best way to get amazing results is to set arbitrary goals without any basis for believing they can be reached. So I set out to search faster than grep by thirty percent.

The first step in any potentially impossible project is, of course, to announce that you are on the verge of succeeding.

I imagine the author of grep, Ultimate Unix Geek, squinting at vi; the glow of a dozen xterms is the only light to fall on his ample frame covered by overalls, cheese doodles, and a tangle of beard. Discarded crushed Mountain Dew cans litter the floor. I look straight into the back of his head, covered by a snarl of greasy locks, and reply with a snarl of my own: You're mine. The aphorism at the top, like the ex girlfriend who first told it to me, is dim in my recollection.

String searching

Having exhausted all my trash-talking avenues, it's time to get to work. Now, everyone knows that without some sort of preflighting, the fastest string search you can do still takes linear time. Since my program is supposed to work on dozens of gigabytes, preflighting is impossible – there's no place to put all the data that preflighting generates, and nobody wants to sit around while I generate it. So I am resigned to the linear algorithms. The best known is Boyer-Moore (I won't insult your intelligence with a Wikipedia link, but the article there gives a good overview).

Boyer-Moore works like this: you have some string you're looking for, which we'll call the needle, and some string you want to find it in, which we'll call the haystack. Instead of starting the search at the beginning of needle, you start at the end. If your needle character doesn't match the character you're looking at in haystack, you can move needle forwards in haystack until haystack's mismatched character lines up with the same character in needle. If haystack's mismatch isn't in needle at all, then you can skip ahead a whole needle's length.

For example, if you're searching for a string of 100 'a's (needle), you look at the 100th character in haystack. If it's an 'x', well, 'x' doesn't appear anywhere in needle, so you can skip ahead all of needle and look at the 200th character in haystack. A single mismatch allowed us to skip 100 characters!

I get shot down

For performance, the number of characters you can skip on a mismatch is usually stored in an array indexed by the character value. So the first part of my Boyer-Moore string searching algorithm looked like this:

```
char haystack_char = haystack[haystack_index];
if (last_char_in_needle != haystack_char)
   haystack_index += jump_table[haystack_char];
```

So we look at the character in haystack and if it's not what we're looking for, we jump ahead by the right distance for that character, which is in jump_table.

"There," I sigh, finishing and sitting back. It may not be faster than grep, but it should be at least as fast, because this is the fastest algorithm known. This should be a good start. So I confidently ran my benchmark, for a 1 gigabyte file…

grep:	2.52 seconds
Hex Fiend:	3.86 seconds

Ouch. I'm slower, more than 50% slower. grep is leaving me sucking dust. Ultimate Unix Geek chuckles into his xterms.

Rollin', rollin', rollin'

My eyes darken, my vision tunnels. I break out the big guns. My efforts to vectorize are fruitless (I'm not clever enough to vectorize Boyer-Moore because it has very linear data dependencies.) Shark shows a lot of branching, suggesting I can do better by unrolling the loop. Indeed:

grep:	2.52 seconds
Hex Fiend (unrolled):	2.68 seconds

But I was still more than 6% slower, and that's as fast as I got. Exhausted, stymied at every turn, I throw up my hands. grep has won.

grep's dark secret

"How do you do it, Ultimate Unix Geek? How is grep so fast?" I moan at last, crawling forwards into the pale light of his CRT.

"Hmmm," he mumbles. "I suppose you have earned a villian's exposition. Behold!" A blaze of keyboard strokes later and grep's source code is smeared in green-on-black across the screen.

```
while (tp < = ep)
          {
            d = d1[U(tp[-1])], tp += d;
            d = d1[U(tp[-1])], tp += d;
            if (d == 0)
              goto found;
            d = d1[U(tp[-1])], tp += d;
            d = d1[U(tp[-1])], tp += d;
            d = d1[U(tp[-1])], tp += d;
            if (d == 0)
              goto found;
            d = d1[U(tp[-1])], tp += d;
            d = d1[U(tp[-1])], tp += d;
            d = d1[U(tp[-1])], tp += d;
            if (d == 0)
              goto found;
            d = d1[U(tp[-1])], tp += d;
            d = d1[U(tp[-1])], tp += d;
          }
```

"You bastard!" I shriek, amazed at what I see. "You sold them out!"

See all those d = d1[U(tp[-1])], tp += d; lines? Well, d1 is the jump table, and it so happens that grep puts 0 in the jump table for the last character in needle. So when grep looks up the jump distance for the character, via haystack_index += jump_table[haystack_char], well, if haystack_char is the last character in needle (meaning we have a potential match), then jump_table[haystack_char] is 0, so that line doesn't actually increase haystack_index.

All that is fine and noble. But do not be fooled! If the characters match, the search location doesn't change - so grep assumes there is no match, up to three times in a row, before checking to see if it actually found a match.

Put another way, grep sells out its worst case (lots of partial matches) to make the best case (few partial matches) go faster. How treacherous! As this realization dawns on me, the room seemed to grow dim and slip sideways. I look up at the Ultimate Unix Geek, spinning slowly in his padded chair, and I hear his cackle "old age and treachery...", and in his flickering CRT there is a face reflected, but it's my ex girlfriend, and the last thing I see before I black out is a patch of yellow cheese powder inside her long tangled beard.

I take a page from grep

"Damn you," I mumble at last, rising from my prostrate position. Chagrined and humbled, I copy the technique.

grep:	2.52 seconds
Hex Fiend (treacherous):	2.46 seconds

What's the win?

Copying that trick brought me from six percent slower to two percent faster, but at what cost? What penalty has grep paid for this treachery? Let us check - we shall make a one gigabyte file with one thousand x's per line, and time grep searching for "yy" (a two character best case) and "yx" (a two character worst case). Then we'll send grep to Over-Optimizers Anonymous and compare how a reformed grep (one that checks for a match after every character) performs.

	Best case	Worst case
Treacherous grep	2.57 seconds	4.89 seconds
Reformed grep	2.79 seconds	2.88 seconds

Innnnteresting. The treacherous optimization does indeed squeeze out almost 8% faster searching in the best case, at a cost of nearly 70% slower searching in the worst case. Worth it? You decide! Let me know what you think.

Resolved and refreshed, I plan my next entry. This isn't over, Ultimate Unix Geek. ■

---

ridiculous_fish is a curious programmer perpetually out of his element. He is channeled by an engineer who currently works at Apple. Read more from fish at ridiculousfish.com

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

You're a Developer, So Why do You Work For Someone Else?

By BRYAN HALES

As a developer, you are sitting on a goldmine. Do you even realize it?

No, seriously, a @#$% goldmine! Never in modern history has it been so easy to create something from scratch, with little or no capital and a marketing model that is limited only by your imagination.

Think about the biggest websites you visit or use on a regular basis: Facebook, Twitter, Flickr, Foursquare, or even Google for that matter — all of them were created by developers who created something from little more than an idea in their head. Was it easy for them? Heck no. But it could only have been done in today's day and age. So why in the world are you sitting there day after day working for someone else?

Yeah, I am too.

So if there are so many amazing opportunities out there, why aren't more developers out there working for themselves? I think there is a pretty common set of excuses that we tell ourselves. None of them are legit!

Myth #1: I don't have any time

This is a common excuse, but one that makes me laugh every time I hear it. Alright, so how much time do you spend watching TV or playing XBox, Wii, Playstation, etc? Maybe just an hour a day right? What about the time you spend playing around on Facebook or Twitter? (Probably just a few minutes here and there, right?) What do you do every day on your lunch break? There's an hour right there.

My point is: an hour here and an hour there adds up! You have time, it's just a matter of what you choose to do with it. If you want to break out on your own, you need to come up with a good idea (one that truly solves a problem) and obsess over it. If you're passionate about your idea, you'll find time. You'll reach a point where it is actually painful to have to work on something other than your idea.

I'm a married 31 year-old guy with three young kids. I work a full-time job and come home to a wonderful wife who, at the end of her day, is at her wits' end with the kids. I consider myself a pretty busy guy, yet I am able to consistently find around 20 hours a week (at least) to work on my idea.

As I write this, I am sitting on a comfy chair across the street from my day job in the café of a Border's Bookstore. I come here nearly every single day, which on its own adds up to 5 hours of pretty productive work per week. No kids running around, no real distractions, just me, my laptop, and my headphones.

In the evening, when the kids have gone to bed and the dishes are washed, I can generally get a good 3-4 hours of work in before going to bed and starting over the next day. I usually give myself a day or two off during the week to keep my sanity and unwind a bit, but with my 20 or so evening hours during the week plus my 5ish lunch hours, I can get some real work done.

Even if you can't afford to quit your day job to pursue your idea (like me), I think you can find time to work on your idea, if you're really passionate enough about it.

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

Myth #2: I can't come up with any ideas

If you're like me when I started, you constantly hear people say stuff like "Ideas are a dime a dozen" and "I'm always coming up with new ideas, but I just don't have the time to follow through." Yet you sit there trying to come up with the Next Big Thing (the next Facebook, the next Reddit, etc) and it seems like all the best ideas are taken. You can't come up with anything that you would consider a home run.

Ask any founder of a large website about how it is today versus how they imagined it would be, and I'd bet they'll laugh. The fact is, they hardly ever start out the way they planned. These sites become huge hits because the founders and owners were smart about adapting and creating features that their users love.

So quit trying to hit a home run and focus on simply getting on base! Create something useful. Something people need, and then iterate over and over and over. Start simple and go from there. If you obsess over the end result (a yacht in the Caribbean on a private island), all you'll ever be is a dreamer. Build something, put it out there, get feedback, and adapt.

Here's what I do when I'm trying to come up with a fun new idea to work on:

❶ Listen to the news (or any talk show for that matter). People love to complain. I see every complaint as a possible idea. My current project, for example, came from a story about the 100th anniversary of the Boy Scouts. I'm a former Boy Scout. I've been a Scout leader. I know their aches and pains, but I had forgotten about them. Listening to the radio and keeping an ear open for opportunities gave me the idea I'm working on now. It's a pretty small niche market, but there is a lot of opportunity there. I get a lot of bad ideas too, but that's ok! Coming up with new ideas is like exercising. The more you keep your ears open for new ideas, the easier it is to come up with new ones and quickly vet them. Find out what people hate, what pains them, and build something that they would be willing to pay for (either directly or through lead-generation, putting up with ads, etc).

❷ What do you love to do? What are you most passionate about? You had better be passionate about what you're thinking about working on, because it will get really tedious and tempting to move onto something else before too long. Make sure, before you begin, that you are ok with working on this new idea of yours 24/7, because you'll need to in order to get it off the ground. One of my passions is scuba diving. I would love nothing more than to live in a world where all I think about is scuba diving. I've got a few ideas for products in that realm that are simmering on the back burner for now.

❸ Keep a backlog. Google Docs is your friend. I have a document that I call "App Ideas." When I get a new idea, no matter how trivial or niche it originally feels, I immediately stop what I'm doing and write it down. I've heard of people keeping notepads by their bed for this same reason. I can't tell you how many "EUREKA!" moments I had in the car on my way home, only to have forgotten them by the time the kids were in bed. It's not that they were bad ideas, it's that I got distracted. They eventually come back to me, but it's frustrating in the meantime. Keeping ideas in a backlog helps you to organize them by legitimacy, add notes and thoughts, and remember them next time you go looking for an idea.

Never start working on a project the same day you came up with it. Let it simmer for a day or two, at least. Make sure that it is worth spending the next few years of your life obsessing over. Don't build it just to see if it people will like it. That will be a complete waste of time. Ask them first. Go read Yes, but who said they'd actually BUY the damn thing? [http://blog.asmartbear.com/customer-validation.html] and come back. Go ahead, I'll wait.

Myth #3: I don't have any money

Who said anything about money? Unless you have come up with an idea that absolutely needs money to get going, which I think should be relatively rare in this Internet world of the Long Tail, you can get going for free. Zero. Zip. Zilch. Be creative about how you get what you need. Barter, trade, consult. Make it a point to spend as little as possible to get things done until you can actually justify spending money you don't have on it. Better yet, don't spend money at all until you've got it coming in from actual customers.

For my current project, I splurged and set up a hosted account at DreamHost for my Django needs. I love it, but I consider it a luxury. I could have built it with Google App Engine for free, but heck, for $100 a year I think I can stomach that. I'm planning on using Chargify at a monthly cost of (you guessed it) free until I get enough customers to justify paying for an account.

You don't need money to get started. If you think you do, and especially if you're a first-time entrepreneur, you should probably think twice.

Myth #4: I don't know how to market/design/etc

This is not a good reason to avoid starting a startup, but I must admit that it is probably the biggest reason people hesitate. As a developer, I am terrified of sales. I hate spending any amount of time on the phone. I don't enjoy thinking of new ways to attract more people to my site. I just enjoy building things. If you find yourself nodding your head, you have one of two choices:

❶ Find a co-founder that is good at what you're not. Focus on what you're good at. If you're a developer, spend all your time listening to your users and building a great product. Sales and marketing are a full-time gig all by themselves. It is extremely difficult to master both worlds. If you have two technical co-founders, you might be able to get by splitting the marketing and sales tasks, but I think you'll find that one of you is better than

the other, and will end up spending more time doing it. Now, just because you're the "developer" doesn't mean that you shouldn't be involved in sales or marketing. Although you'll get the most bang for your buck by playing to your strengths, you should also know exactly what is involved in the sales, marketing, or PR side of things. That will prevent you from ever saying to yourself "Man, why can't John ever bring in any real customers? Why do I feel like I'm doing all the work?" When you realize how hard marketing and sales are, you'll appreciate it more. Get your hands dirty! Step up! Conversely, sales or marketing-savvy co-founders should spend some time at least reading through the code. Give them a chance to contribute a little. At the very least, they might consult with you about a new feature before selling one that doesn't exist if they know how hard and time-consuming your job is as well. Take the magic/black-box aura out of the equation and get your hands dirty!

❷ Step up and learn how to do it. This will mean that you will need to set aside your code for awhile and learn how to market effectively or essentially become a Sales/PR person. It takes time, so don't give up! The good news is that what works for one company or website will not necessarily work for another. "What," you say? "That sounds like bad news." Look at it this way: The worst you can do is fail. I say that tongue-in-cheek but it's true. If you fail at a marketing campaign, so what? TRY again some other way. Add it to your list of failures and move on. Learn what you can from books, forums, websites, how-tos, etc, and then go out and EXPERIMENT. You don't have to have money to experiment either. Be creative and resourceful. You need to learn about what works for your company, not someone else's. Take what lessons you can learn from others and try something.

Myth #5: I need a steady income — I can't quit my job!

This may be more of a reality than a myth, but it is no reason to continue with the status quo. Do you really want to work for someone else every day, on their terms, for the rest of your life? No? Well that's going to require some sacrifice. Of course, you know that, otherwise you wouldn't be reading this article!

If it is even slightly possible, the best thing you could ever do would be to quit your job and focus 100% of your time on your startup. Doing so forces you to focus on quality and making something people are willing to pay for. The need to pay bills and buy food is an incredible motivator.

If, like me, you have young mouths to feed and quitting just isn't an option, you can still find time — it will just take longer. "See Myth #1: I don't have any time."

Great startups don't happen overnight. They take time. It can take years to really gain some traction. Don't give up!

Once you have a decent working prototype built, go back to the people who told you it would be a good idea (you did do that in the first place, didn't you?) and get feedback. I have found that this is a great source of encouragement. You'll probably get some haters, but consider that a good thing! If people are passionate about your project, then you may have hit a nerve. Take in their criticism and improve. The last thing you want is a bunch of people telling you something is a great idea, because they don't want to offend you. What you end up with in that situation is a mediocre product that nobody really cares about.

Once you start gaining some traction and real users, consider getting your project funded. Ask friends and family to invest or talk to an angel investor. If you can't convince them to fund you, that doesn't necessarily mean your idea sucks, it just means you need to refine it and get more users. If you can get and retain users, then you're obviously on to something. In this world of the Long Tail, you don't have to have a massively funded or mainstream project to make money!

Myth #6: I can't find a partner

One of the biggest reasons startups fail is because of bad partnerships. Infighting or co-founders who are not pulling their fair weight can kill your idea faster than anything else. It is extremely important that you pick a co-founder who is as passionate about your idea as you are.

Don't expect someone to be as passionate about your idea as you are right off the bat. You have had a lot more time to think and dream about it than they have. Criticism and playing "devil's advocate" should be welcomed when discussing an idea. Do you really think you speak for everyone? You should actually welcome dissent, as long as it is constructive criticism.

Where can you find a good co-founder? The best place to look is among people you already know. There is a much lower risk of a personality clash if you already know them and their working habits and passions.

Because starting a startup is hard work with little payoff initially, you need someone who understands and appreciates this. Take part in communities like Hacker News or Founders Mix to find people that think the way you do.

If you have a community like Gangplank near you, go hang out. Learn from people there and don't be afraid to share your idea with everyone you come in contact with. Don't try to guess what people want, ASK them! It's silly to walk around afraid to mention your idea to anyone because "they might steal my idea." Ideas are a dime a dozen. If you're afraid that they can execute your idea better than you, then you have bigger problems.

Now, get out there and build something people want! ■

Bryan Hales is a C# developer by day and a Django hacker by night. He has been in love with the entrepreneurial spirit and attitude since he was a young boy. Whether or not you actually end up running a startup of your own or not, he is a firm believer that the Startup Culture can offer businesses of any size considerable advantages. He writes about his entrepreneurial thoughts and experiences at http://www.intermittentintelligence.com

Advice to Aimless, Excited Programmers

By JAMES HAGUE

I OCCASIONALLY SEE MESSAGES like this from aimless, excited programmers:

> *Hey everyone! I just learned Erlang/Haskell/Python, and now I'm looking for a big project to write in it. If you've got ideas, let me know!*

or

> *I love Linux and open source and want to contribute to the community by starting a project. What's an important program that only runs under Windows that you'd love to have a Linux version of?*

The wrong-way-aroundness of these requests always puzzles me. The key criteria is a programing language or an operating system or a software license. There's nothing about solving a problem or overall usefulness or any relevant connection between the application and the interests of the original poster. Would you trust a music notation program developed by a non-musician? A Photoshop clone written by someone who has never used Photoshop professionally? But I don't want to dwell on the negative side of this.

Here's my advice to people who make these queries:

Stop and think about all of your personal interests and solve a simple problem related to one of them. For example, I practice guitar by playing along to a drum machine, but I wish I could have human elements added to drum loops, like auto-fills and occasional variations and so on. What would it take to do that? I could start by writing a simple drum sequencing program – one without a GUI – and see how it went. I also take a lot of photographs, and I could use a tagging scheme that isn't tied to a do-everything program like Adobe Lightroom. That's simple enough that I could create a minimal solution in an afternoon.

The two keys: ❶ keep it simple, ❷ make it something you'd actually use.

Once you've got something working, then build a series of improved versions. Don't create pressure by making a version suitable for public distribution, just take a long look at the existing application, and make it better. Can I build an HTML 5 front end to my photo tagger?

If you keep this up for a couple of iterations, then you'll wind up an expert. An expert in a small, tightly-defined, maybe only relevant to you problem domain, yes, but an expert nonetheless. There's a very interesting side effect to becoming an expert: you can start experimenting with improvements and features that would have previously looked daunting or impossible. And those are the kind of improvements and features that might all of a sudden make your program appealing to a larger audience. ■

James Hague is a recovering programmer who now works full time as a game designer, most recently acting as Design Director for Red Faction: Guerrilla. He's run his own indie game studio and is a published photographer.

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

Agile Ruined My Life

By DANIEL MARKHAM

I READ THE REPLY to my comment on a popular hacker board with sadness:

> *(disclaimer: Agile consultants ruined the software group I work in.) Making good software is hard, and anyone claiming to have a magical process that guarantees good software is selling snake oil. I can appreciate your wanting to make a buck, but would also seriously appreciate it if you could find some other industry besides software development to go screw up*

Reminded me of an email I received back in May:

> *[We] started working on [agile technique X] when [author]'s [famous book] was just a draft. I was on that project and worked on Agile Projects for a decade. (Next time you meet [famous guy], ask about me, I just finished reviewing his forthcoming [another famous book]). I am a founding member of the Agile Society of [place] and have organized conferences on Agile. I've attended XP Conf as well. I've probably worked in more agile projects than you ever have (not that it particularly matters). So let us first dispense with the notion that your notion of what constitutes "true" agile and its scamsters is somehow the only standard....*
>
> *Do you deny that the whole Scrum Master idea is a scam within the Agile Camp?*
>
> *Scamster? Ron Jeffries the guru/ founder of Agile couldn't write a Sudoku implementation with his favorite technique "TDD" and refactoring over five weeks. Fraud.*
>
> *Robert Martin (another "guru" and agile consultant) claims that any code not written with TDD is "stone age" code including such things as Unix and such people as Norvig and Linus and Zawinski who've built more code than he can dream of. Dalke poked holes in his TDD "kata" which never got answered Fraud.*
>
> *I could go on and on. And these are the gurus. But that isn't the point. i *saw* "agile consultants" evolve from some naive but well meaning people (like Kent) to scamsters like X and co and tose are just at the top. Practically every single "Scrum MAster" is a fraud. The more intelligent among them admit that two days of listening to a higher level shyster teach nothing and it is just a signal to dumb managers to improve their chances of getting a project. Yet they go along. That in my eyes is a scam like chiroproctors or reiki people claiming to be doctors. Agile was amovement founded by scamsters and propagated mostly by scamsters.*

I've had many such conversations over the years. There are some seriously pissed off people about Agile out there. Why? Isn't agile supposed to be warmth, apple pie, motherhood, goodness and all of that? Why so much anger?

The easy answer – and the answer most agile-lovers would give – is that these folks are simply non-hackers. Bad attitude, poor skills, interpersonal conflicts – the reasons are many and diverse as to why a small percentage of folks are just going to get ticked off at anything you try to do.

I don't accept that. Or rather, while it may be true, it is also an excuse for non-action. I view every piece of feedback as a cause for some kind of action.

And the thing is, it's not just the people who are being trained. I've done my fair share of complaining about various pieces of agile, and I've seen many other coaches – in private– grumble and complain as well.

So it's time to get honest. Take a good look at ourselves.

Here are the problems I see and hear about:

- Fake success stories - People think they can take some lame project that's mostly done, apply a little agile, then proclaim how great it was? Come on, folks, this isn't fooling anybody. Everybody knows exactly what it is: propaganda. Making it worse, many times the experts brought in are the last to know what a pointless photo-op exercise it was, leading them to "learn" incorrect things from the experience as well, then "sharing" that knowledge with new teams, continuing the cycle of crap.
- Trainers who can't do the work - I have good friends who teach agile and haven't coded or led a team in years, so to them I apologize. But if you're going to train something, you should be able to do it. And I mean do it to a very high level of expertise. An agile coach should be able to code, perform analysis, manage the project, test – anything that needs doing on a project. If she can't, then how can you talk to her about your particular situation? If your agile trainer was a BA last week, or never slung code in his life, or is a professional trainer, or – let's be brutally honest – is making less than the members of the team are, you've got a dud. It seems like common sense but it bears repeating: you can't train something you haven't done. And "done" means a bunch of times, not just on the pilot project. I had a company once that wanted me to train several people to be agile coaches – people who never knew agile before I walked in the door two weeks before. Hell, if I could do that I'd be printing money, but it doesn't work like that. Does anybody go to school to be a famous baseball coach? Or do they learn to play baseball first and then only some of them realize that they have a talent for coaching? Nine women can't have a baby in one month, no matter how much you wish it were so.
- Inflexibility on the part of adherents - I worked with a lady who wrote an article asking "why are we complaining about Scrum teams not succeeding when they're really not doing scrum?" This attitude – that there is a list of things that must be perfectly done and failure is a result of not doing them – is basically religious in nature. You can never do enough. If the team fails? Well, it wasn't agile enough. It's nonsense, that's what it is. Lots of great agile teams fail. And lots of teams who are not agile do very well.
- "Feel good" agile - One of my friends went to an agile conference. She told me she left one class because it was about "the use of haiku in team-building." While I love poetry, seems a bit fluffy to me (and to her). I have several friends who, god love them, are hippies. It's all bunnies and floaty clouds and harmonics and karma. These things may have an important part in life, but I don't want to sing Kumbaya, I want to have a fun and productive team. Don't get me wrong – I love unusual and off-the-wall techniques. But the agile community has at times embraced the far fringe of wackiness too. It's hard enough getting extremely detail-oriented analytical people to stand up and talk to each other every day. Getting them to put on a puppet show for their showcase is just a bridge too far. We need to tone it down.
- Magic Bullet Syndrome - One of the first things they tell you in Scrum class is that Scrum is not a magic bullet. Then they spend the rest of the time telling you how it's the best thing since sliced bread. We've all met and worked with the guys who already have the answer – you just need to ask the question. The solutions have already been determined for whatever problem you might have. These people are extremely annoying. It's like talking to a wall. Bad, bad. I knew a guy once (another famous guy) that would love to stand up and give an impassioned plea for just doing scrum. Whatever the problem, whatever the actual situation, you could count on him to bloviate about Scum. Not only ineffective, but highly annoying. You don't know whether to laugh or cry.
- Reversal of team dominance - I know a lot of guys who teach agile. Sadly, many of them impose agile on teams, not train them. You come in with a big stick, then proceed to beat people with it until they "conform." The dynamic is backwards – the outsider is somehow in charge instead of the team. One guy (famous author again) basically put it like this to me when I told him the team wasn't succeeding: I'm here to demonstrate certain practices and to show that they work, not to just stop everything and attend to what the team is dealing with today. Sadly, upper-level management encourages this kind of behavior. Many clients will ask me to provide a schedule an a checklist for how I'm going to make their teams agile. I tell them look, I can provide the information, and I can coach the team as it

> **“One of the things I hate most about agile is when management decides to “be” agile, only they don’t want to change anything.”**

works the problems, but the problems are going to be people problems, not technology problems. And guess what? People don’t respond very well to being treated like machines.

- Cargo Cult Agile - There are a lot of teams doing cargo cult agile out there, also called theater agile. It’s where everybody knows their lines, the rituals, and where to stand and how to act. It’s like an orchestrated pagent. It’s an awful, lifeless thing. Blech
- Non-answers to questions - Can agile work with distributed teams? It depends. Can we use fortnights to estimate projects? It depends. Can agile work with embedded software? It depends. Argghhh! Everything depends. Sometimes no matter how hard I try to be forthcoming, honest, and direct, I end up sounding like those guys from The Matrix the first time you watched it. They said a lot, but it didn’t really mean much. It all sound like just so much gobbledygook. I hate giving advice like that. I know folks hate hearing it.
- Conflicting Advice - Can you architect and design before you code with agile or not? Can you have requirements? Can you work on requirements ahead of the sprint you are in? Can you roll-up multiple projects into usable program management metrics? I could list a dozen more questions which have multiple answers, depending on who you talk to. One guy says do it this way. Another guy says do it that way. It’s enough to drive anybody nuts.
- Scamsters - I spoke at an Agile conference back in 2009. One of the first things I said to the audience was “I don’t read agile books. They are a waste of time.” Wow! You could almost hear the groans throughout the crowd! But I’m serious: 99% of agile books out there are just people telling stories about stuff. Stories are great – love to hear them. But I can’t trust the authors of most of these books to tell honest stories and learn honest lessons from them. Instead they have a theme, an argument, a point-of-view. And everything in that book is going to support that theme, that point of view. Heck, it’s just good book-writing. The problem is, real life doesn’t have a theme. Or if it does, it would be amazingly incredible and preposterously improbable if your book matched up with what was going on with my organization. The early agile books were so funny that I couldn’t read them – I always started laughing too much. When Bob Martin started trashing developers who didn’t use TDD I realized that many “agile experts” were jumping the shark. Yes, there are a lot of folks selling you things you don’t need by convincing you that you need it. A fair word for that is “scamster.”
- It’s the same, only different - One of the things I hate most about agile is when management decides to “be” agile, only they don’t want to change anything. So then you’re teaching a team that they are in control, that they are responsible for important decisions about how much work they can do in each iteration and how to do it – only they aren’t. This destroys morale faster than anything. A while back I turned down teaching TDD to a team. Why? Because somebody up above had decided the team was doing TDD, not the team. The class would have been three days of me trying to share things that the team had no desire to hear and wasn’t going to practice. A lot of other coaches – the vast majority, probably, would have taken that gig, but I’m not going to be part of the problem. Courage isn’t just for teams.
- Little Gold Star Syndrome - A two-day class and a little gold star, or your name on some website, doesn’t mean jack squat in terms of what you can do. Let’s just be honest about that. The training might be great, but the idea that getting a little gold star sticker on your head is going to make you significantly better is bogus.

When training agile, one of the first things I do is go over definitions.

What’s Iterative Development?
What Incremental Development?
What’s Scrum?
What’s Agile?

The answers – both the ones I give and the class’s – are interesting.

Iterative development is doing things in iterations. Little bits of work done over and over again. Agile is big on time-boxes, but iterations can be done based on features too. The idea is that you do everything you need to do to deploy. Then you do it again. Over time the product gets better and better and the team begins to have experience in all phases of development.

Incremental Development is doing things in little atomic pieces, called increments. Say you want a checkbook

> **“You don’t just read a book or take a class and suddenly you are agile.”**

program, so increment 1 might be logging in. Increment 2 could be writing a check, etc.

So far, so good. Most folks think that iterative and incremental development are good ideas. If not, then welcome to 2010. There are other ways of doing things, sure. But most folks are already at this point.

So what’s Scrum? It’s a standardized version of project management tools for iterative and incremental development, that’s what. It has a board, a test, a class. It’s a monolithic thing. When we talk scrum we have concrete terms and concepts to discuss (like them or not, separate subject).

Our final question: what’s Agile? Usually a couple people have ideas. “It’s TDD” one might say. Another might say “There’s a manifesto I think”

After a long pause I tell the class what sounds like a joke but isn’t.

Agile has no definition.

Nada. Zip. Bupkis.

There’s no standards board, there’s no test, there’s no approved workbook, there’s no checklist.

Agile is nothing like Scrum. Personally, I think that’s a good thing.

Agile is a set of best practices around running iterative and incremental development teams. It’s a marketing term. Sure, there’s a manifesto, and there are experts (I’m one of them), and there are conferences, and books, and classes, and god knows what else. But it’s just best practices.

It’s based on three things: ❶ principles not practices, ❷ attention to people, and ❸ always be adapting

To some, this might be so fuzzy as to mean nothing. If so, I apologize. I can assure you that there really is a structure and line of progress to learning agile. I can also assure you that teams that “get it” are happier and produce a lot more than teams who don’t.

But it IS an art, not a science. You don’t just read a book or take a class and suddenly you are agile. It’s more like playing jazz piano. You learn a bit, you do a bit, you take an honest inventory of what works and what doesn’t, then you learn a bit more. And so on. It’s the doing , the reflecting, and the adapting that count the most. You don’t learn to play the piano by watching a film of somebody else playing, reading a book about it, or going to a conference. And you don’t learn by making yourself into a robot, following a series of rules without exception. Would you try to play the piano by dressing up like a pianist, renting Carnegie Hall, and simply acting as much like a great piano player as possible? Yet every day some poor schmucks are sitting in a stand-up that lasts for an hour and is more of a brutal daily status report than something collaborative. And we call that agile.

The commenter from yesterday went on to say that he was working in a development group that was happy and productive. Then they were bought out by a larger firm who decided to “do agile” on them. Productivity went down the tubes, morale suffered, and people were told to adapt or get lost.

Iterative and incremental development isn’t for everybody. Lots of teams do things completely ad-hoc. Lots of teams are happy with waterfall. Lots of folks just don’t care to change. These are all good reasons why agile might be a bad idea.

My standard for what agile isn’t universal, sure. but I’m very happy teaching best practices for iterative and incremental development. You can call that agile, you can call it Joe. Whatever it is, helping people see things and try things they haven’t seen or tried before – and then letting them decide whether it’s working for them or not – is a pretty good business to be in. But there are a LOT of problems in this business, and ignoring them won’t make them go away. Over time there can be an us-versus-them attitude that sets up between any two groups of people. We must always be on guard for this. If you’re not a servant to the team, you shouldn’t be in the room. That’s just as bluntly as I can put it. The problems listed above are owned by all of us, and it’s our job to make sure we address them as best we can. ■

Daniel Markham is the principal partner of Bedford Technology Group, a consulting firm that trains teams in large organizations to run like startups. When he’s not consulting with large companies, he’s a serial inventor and small team/startup junkie who has created dozens of various websites and apps. Daniel programs in most all major languages and database platforms, although he currently is excited about working in F#. He lives with his wife and 2 kids in rural Virginia.

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

On: Product For People Who Make Products For People

From REGINALD BRAITHWAITE (raganwald)
I hear a lot of "programmers don't do good UI" as well as "marketers dictate bad UI" in my travels. I used to try to work out some sort of theory about which statement is true, and why. But then I experienced a revelation, Sturgeon's Revelation:

90% of everything is crap.

Therefore, if handed ten UIs designed by programmers, nine will be crap. If handed ten UIs designed by marketers, nine will be crap. Perhaps there is a characteristic way in which the nine programmer crap UIs are crap, but the observation that most programmer UIs are crap is not insightful and doesn't magically justify the idea of turning UI design over to product management.

On: Agile Ruined My Life

From RAPHAËL AMIARD (Raphael_Amiard)
As a young software developer, what really bores me with Agile, is the name, the shiny box you put things into, where it should just be named "Good practices for software development." It's the mentality of selling things as products, with some kind of prebuilt ideology and aesthetic built along with the core, that really makes me run far far away.

I don't want to be sold a product. The fact that it led people to try new ways of developing software, be it TDD or pair programming or whatever, is good, but heck, just give me the core idea, remove the gift wrap, and go away. I don't want nor need some kind of new age manager coach.

Anyway, the article seems to be making this very point in some way, but then, why the name agile? Well for marketing of course. So, while i sort of agree with the article, well I'll just be far away looking, thanks.

On: Advice to Aimless, Excited Programmers

From DANIEL KROL (orblivion)
It's funny because some of us have the opposite problem. Too many ideas, but not that much interest in learning new languages for their own sake. So when it comes down to implementing a new idea, we want it done quickly. We don't want to take the time to try to figure out how to make it in Haskell, and just revert back to Python.

On: How Do I Write So Much

From JACQUES MATTHEIJ (jacquesm)
So, is this one of the crappy ones or one of the good ones? ;)

Agreed whole heartedly though, if you are a "producer" there will be tons of stuff that is not fantastic but that might be useful to somebody.

The funny thing is that it is unpredictable, what will be appreciated and what not. Sometimes I fire off a 10 minute blog post and it gets retweeted for days or even weeks after, and sometimes I work for hours and hours on something and nobody cares.

I see the "lower grade stuff" as taking a break from the other stuff whilst still keeping busy. Sooner or later you find yourself engaged with more interesting things again, if you "broke the routine" just because you're not doing anything worthwhile you'd find your source of inspiration dried up pretty quickly.

So keep busy, by all means, and fail often, looking forward to the gems. Like this one: http://sebastianmarshall.com/?p=95

On: You're a Developer, So Why Do You Work For Someone Else?

From NIR YARIV (nir)
I'm happy for the author and wish him success. But the assumption that being a good developer can make you rich is misleading.

Reading PG may create the impression that building a successful company is an engineering-like process, deterministic and repeatable. It is not. It's a chaotic process that cannot be reliably planned. Thinking "Zuckerberg coded a PHP app, I can code a PHP app" is like thinking "That old lady bought a lottery ticket. I can buy a lottery ticket.."

Building a smaller business that supports a few people is a different story (and a worthy goal unto itself). But that's not "sitting on a goldmine."

EDIT: Just wanted to add I have huge respect for this guy, building a business while supporting a family. My beef, such as it is, is with simplistic picture of startups often painted in HN. The actual people giving it a go, you have to respect.

On: Staying Healthy and Sane At a Startup

From DAVE GALLAGHER (dgallagher)

Exercise is basic necessary maintenance required by the human body. You either do it and have a happy/healthy body, or you don't. Eating healthy is just as important, separated into two categories: overall caloric intake, and nutritional content.

Car analogies are a dime a dozen in computing, but they apply here as well. Not exercising is like changing the oil in your car every 10,000-15,000 miles, instead of every 3,000 - 6,000. Your car is still going to last years, but its lifespan will be shortened, and it's going to run poorly towards the end of it. The nice thing about a car is you can repair it, or buy a new one. Repairing a human is tricky, and you definitely can't buy a new one.

I'm absolutely baffled by those who put their careers or money at a higher priority than their own physical health. You really want to be rich and famous with a crappy body? Is type 2 diabetes, along with likely amputations, blindness, and erectile dysfunction, your thing? Looking forward to clogged arteries and heart disease? What about stroke, wiping away your ability to control your own body, or even being able to think or speak? Etc, etc...

I don't intend to be mean, but many American's simply don't prioritize their health high enough. People seem to have every excuse in the world not to do it, except for a good one.

True story. My brother is enrolled for his doctoral in physical therapy. He dissected cadavers (donated human corpses) during one of his classes. My family and I went out to visit, and he was able to let us look at one.

My father, who is about 30 pounds overweight in his late-50's, hasn't really cared about his health. He eats too much high-saturated-fat ice cream, puts cream in his coffee, likes cookies with lots of butter in them, etc... I've been trying to get him to eat healthier and exercise for years to no avail.

Well, my brother was having me hold/feel the heart from the cadaver (it was already cut out of the dissected body). I was squishing some of the arteries with gloves on, and my brother said "Try squishing this Coronary Artery. Sometimes it might be crunchy from heart disease."

So I did, and WOW! It was ROCK SOLID! So much calcium and plaque had built up inside this persons heart that it completely clogged the artery. It was as if there was a pebble-sized rock inside of it.

Of course, I forced my father to put some gloves on and feel it for himself. Well, that scared the SHIT out of him! These last few weeks since, he's made a decision to stay away from high-fat foods (and has been doing so - non-fat ice cream now, skim milk in coffee, etc...). He's also putting together an exercise room.

It looks like he's in the right mindset now, which is a very good thing! Sometimes the dagger of death hanging over your head is the best motivator. :)

On: Most Common Words Unique to 1-star and 5-star App Store Reviews

From TOM DARROW (lotharbot)

The words that leap out at me from the one-star list are "actually" and "says," words indicating that real behavior differed from expected/advertised behavior. The app says X but actually does Y, and therefore sucks. This strongly underscores the value of consistency between what your customers think they're getting and what you actually deliver.

For links to the posts on Hacker News, visit *hackermonthly.com*. All comments are reprinted with permission of their original author.

Dream. Design. Print.

MagCloud, the revolutionary new self-publishing web service by HP, is changing the way ideas, stories, and images find their way into peoples' hands in a printed magazine format.

HP MagCloud capitalizes on the digital revolution, creating a web-based marketplace where traditional media companies, upstart magazine publishers, students, photographers, designers, and businesses can affordably turn their targeted content into print and digital magazine formats.

Simply upload a PDF of your content, set your selling price, and HP MagCloud takes care of the rest—processing payments, printing magazines on demand, and shipping orders to locations around the world. All magazine formatted publications are printed to order using HP Indigo technology, so they not only look fantastic but there's no waste or overruns, reducing the impact on the environment.

Become part of the future of magazine publishing today at www.magcloud.com.

25% Off the First Issue You Publish

Enter promo code **HACKER** when you set your magazine price during the publishing process.

Coupon code valid through February 28, 2011.
Please contact promo@magcloud.com with any questions.

MAGCLOUD