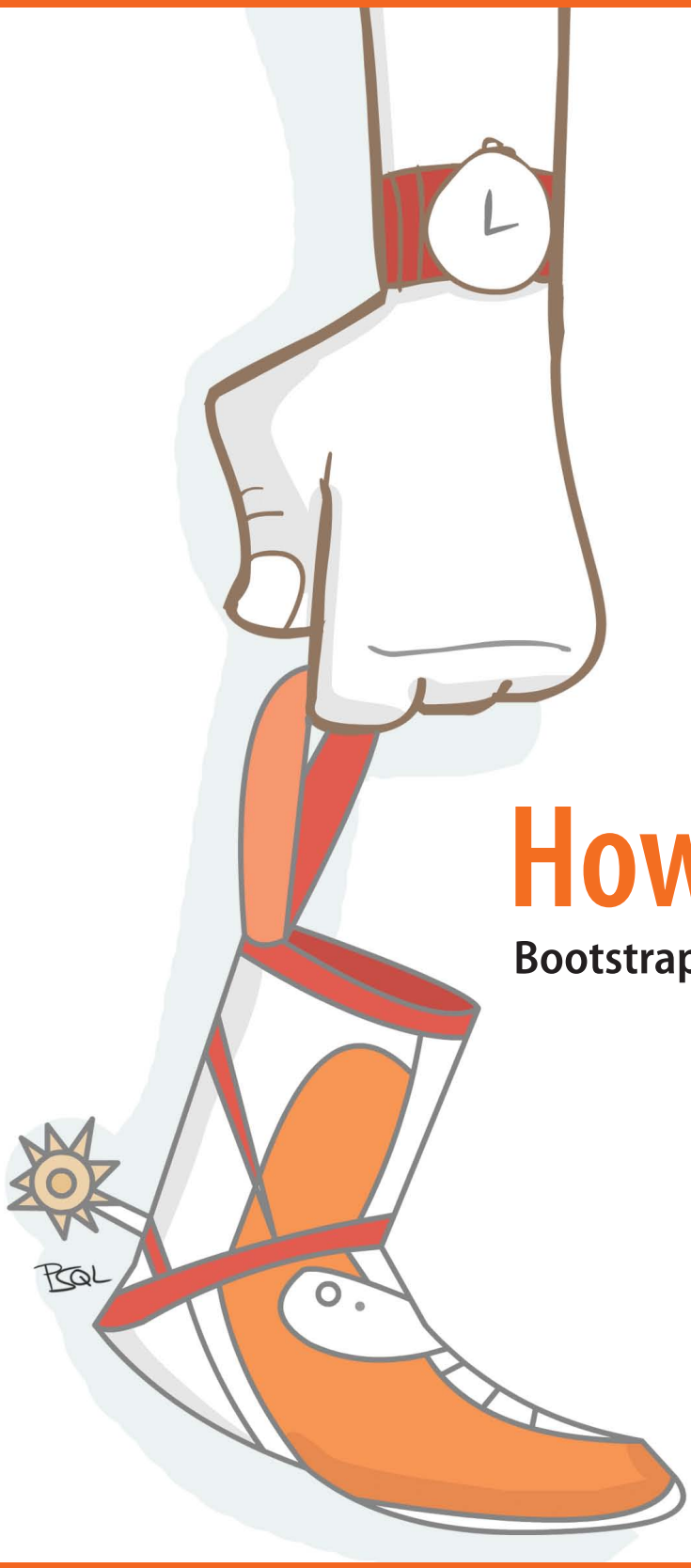


How to Bootstrap

Bootstrap Your Company to Profitability!

HACKERMONTHLY

Issue 2 July 2010

Curator
Lim Cheng Soon

Contributors
Spencer Fry
Matt Welsh
Joey Devilla
Geoffrey K. Pullum
Mike Taylor
Jeff Atwood
Zack Linford
Jacques Mattheij
Zack Hiwiller
Bruce Schneier
Dominic Szablewski
Jakob Nielsen
Rafael Corrales

Proofreader
Ricky de Laveaga

Illustrators
Jaime G. Wong
Pasquale D'Silva

Printer
MagCloud

Advertising
ads@hackermonthly.com

Rate Card
hackermonthly.com/ratecard

Contact
curator@hackermonthly.com

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

Curator's Note

I'M OVERWHELMED BY the overall reception from the launch issue of Hacker Monthly. It sold more than two hundred copies (my goal was a hundred), has been downloaded more than ten thousand times, and email subscribers have more than doubled (3,900 and counting). Best of all, lots of readers sent in their form of support, whether it's a simple email, suggestion to improve, donation, or offer to help. Thank you all.

In this issue, I'm especially grateful for the help of the excellent proofreader, Ricky and the incredibly talented illustrator, Jaime.

A new section has been added in this issue, called Hacker Comments. We created Hacker Comments thanks to suggestions by our readers, who made a strong point that the most interesting thing about Hacker News is the comments. Indeed. — *Lim Cheng Soon*

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

Cover Image by Pasquale D'Silva.

Contents

Illustration by Jaime G. Wong. Check out his work at *http://retrazos.pe/*.

FEATURES

PROGRAMMING

STARTUP

SPECIAL

How to Bootstrap

By SPENCER FRY

In my 10+ years of running Internet companies, I've never raised a single dime, yet I've still gone on to sell three profitable companies and am currently on my fourth, Carbonmade. Bootstrapping is something I'm very familiar with, so I've gathered together some thoughts that should provide you a step-by-step process of going from idea to product to profitability. I have nothing against raising money — angel or venture capital — it's just not the process I'm most familiar with. How to bootstrap goes hand-in-hand with how to run a lean startup, so expect some crossover below.

Idea Generating

Idea generating is only slightly different when you're bootstrapping than when you're looking to raise money. The only important difference is: if you're planning to bootstrap your idea must have built-in revenue generating functionality from the get go. Building Twitter is off the table. You can't wait to hit scale before turning on the revenue features. That's why ideas around Software as a Service (SaaS) are so effective for bootstrapped companies, because you only need one customer to reach revenue — and, with inexpensive hosting costs, probably only a dozen or two to reach profitability.

Bootstrapped companies can't afford to wait around to reach a network effect. You need to start generating dollars as early as possible so that you can quit your day job or put a stop to the draining of your bank account as soon as possible. Bootstrapping startups don't have the luxury to wait around. So when generating an idea for your startup, toss out everything that doesn't involve charging a fee for at least some of your clients. Leave the ad revenue and crazy business model revenue streams to the startups with venture funding. That's just not your game to play.

Team Building

You can either come up with the idea first or the team first. I think it's fine to do it in either order, but it's probably best to come up with the idea before the team. Then you can build a team around the idea. When bootstrapping, you need to find a team that's willing to work for nothing and spend their off hours with you, so finding these types of people can take some searching. You're far more limited in your choices.

The worst thing you can do is work with people who can't comprehend the idea of bootstrapping. You need to work with people who understand that their nights and weekends are going to be fully dedicated to building a product. They'll be working two jobs, not one. You need to explain to everyone you depend on how a bootstrapped company works: Revenue generation is slow at first, though steady, and it could take a year or more of hard work before they can quit their other job and work full-time

> **"Leave the ad revenue and crazy business model revenue streams to the startups with venture funding."**

on the company. But the advantage here is that after a few months off the ground you'll have a clear sense of how soon that day can come. Another advantage of a bootstrapped company on the SaaS model is that it's really easy to calculate your cash flow.

It goes without saying that the people you work with should have complementary skills to your own, but the bootstrapper's "slow but steady" mindset is just as important to the health of your company. You'll find a lot of people may not be comfortable with this approach. Weed those people out as co-founders when you're bootstrapping a company. A one and done approach won't work here.

Off Hours

Almost every bootstrapped company begins as an off-hours tinkering project. That's true of Carbonmade, which Dave built for himself first; that's true of TypeFrag, which I built over the course of a week during my sophomore year in college; that's true of 37signals' Basecamp, true of Anthony's Hype Machine and lots of other companies.

The good thing about bootstrapping is that you don't need to spend a single penny outside of server costs and you can even do most things locally before having to pay any money on a server. Your biggest expense is time, and that's why off hours are so important.

Consult on the Side

The way we started Carbonmade, the way 37signals started, the way Harvest started, and many other startups too, was by first running a consulting shop. We ran a design consulting company called nterface that Carbonmade grew out of. It's great, because the money you're bringing in through client work tides you over while you're waiting for your startup to grow.

Carbonmade was live for nearly 18 months before we started working on it full-time. During those first 18 months, we were taking on lots of client work to pay our bills. The great thing about consulting through the early months is that you can take on fewer and fewer jobs as your revenue builds up. For example, you may need a dozen large projects during the first year and only two or three during the second year. That was the case for us.

I know of other successful bootstrapped companies that during the first year would take on a single client project for a month or two, charging an appropriate amount, and that would give them just enough leeway to work on their startup for two or three months. Then they'd rinse and repeat. They did this for the first year and a half before making enough money to work on their startup full-time.

There's No Need to Rush

When you're bootstrapping there's no rush to get things out the door, even though that's all you hear these

> "If you're too worried about getting off the ground quickly, then you're bound to make a mistake."

days. I know people talk about iterating quickly, and that's all well and good, but when you're bootstrapping and not meeting anyone's deadlines but your own you can take your time to better perfect your product before every release. In my opinion, you should strive to be more Apple-like and really think things through. If you don't take money from an investor who will demand quick new product releases, you can take the time it needs to perfect things.

The first few iterations of your product are everything, and bootstrapping through this beginning phase can allow you to take your time and think through everything. If you're too worried about getting off the ground quickly, then you're bound to make a mistake.

Building Organically

Bootstrapping a company allows you to grow it organically. We at Carbonmade always refer to this as incubating your project. We like to release something, let it sit, feel and gauge the reaction, and then move on from there. You don't have this kind of freedom when you're not bootstrapping, because you're desperately trying to ramp up as quickly as possible.

I've heard stories of companies acting too quickly on initial feedback only to undermine themselves going forward because the feedback was from the wrong user group. For example, if only web designers had given us feedback in the early days of Carbonmade, demanding more precise tools for editing the look and feel for their site, we would have never realized that our market is far more broad: the masses of creative people who don't have a build-it-yourself skill set. We would have limited Carbonmade to a smaller group of people and never have gotten as big as we are today.

Making That First Dollar

Bootstrapping is all about making that first dollar. When I launched TypeFrag we didn't get any sign-ups for the first week and this got us very worried — my partner and I almost threw in the towel — but about five days into it we got our first bite. Then another. Then three the next day. And more and more. Sign-ups began to pile up well beyond what we had anticipated.

All this money coming in meant we could begin to lay out our plans. If no money had come in, we would have had to drastically change directions. Revenue validated our idea, and as every dollar came in we got a better sense of our cash flow and could plan the future development of TypeFrag more accurately. We were able to quickly figure out that people wanted PayPal, so we add that and saw even more money come in. Your first dollar validates your product, your business model, and everything else.

When Investors Come A Calling

As soon as you make that first dollar, investors are going to start making inquiries. That's a good sign! It means you're doing something right. They're not scary guys and most of them are really nice and great people to meet with! Even Jason Fried, the man who is well known for scorning investors, says in 37signals' 13th podcast that it may even make sense for your bootstrapped company to take investment after you've gotten off the ground. I completely agree, as long as you know exactly how you're going to put that money to use. Furthermore, the outcome you anticipate you'll get from taking money needs to be well beyond what you anticipate doing without it.

My advice: Consult with a select few people you really trust who aren't tied too closely to your company and see what they have to say. Try and find someone who has raised money before and had a successful outcome or two. Share everything with them and see if taking that $2.5 at a $10m valuation makes sense. Can you put that $2.5m to use to make your company worth at least 10x more than it's worth today in three to five years? ■

Spencer Fry is the co-founder and CEO of Carbonmade, handling day-to-day operations, accounting, legal matters, customer service, marketing, advertising, and "everything else" that's not design or code. Carbonmade is the easiest way to display and manage your portfolio online, with over 225,000 members.

Reprinted with permission of the original author. First appeared in *http://spencerfry.com/how-to-bootstrap.*

catn

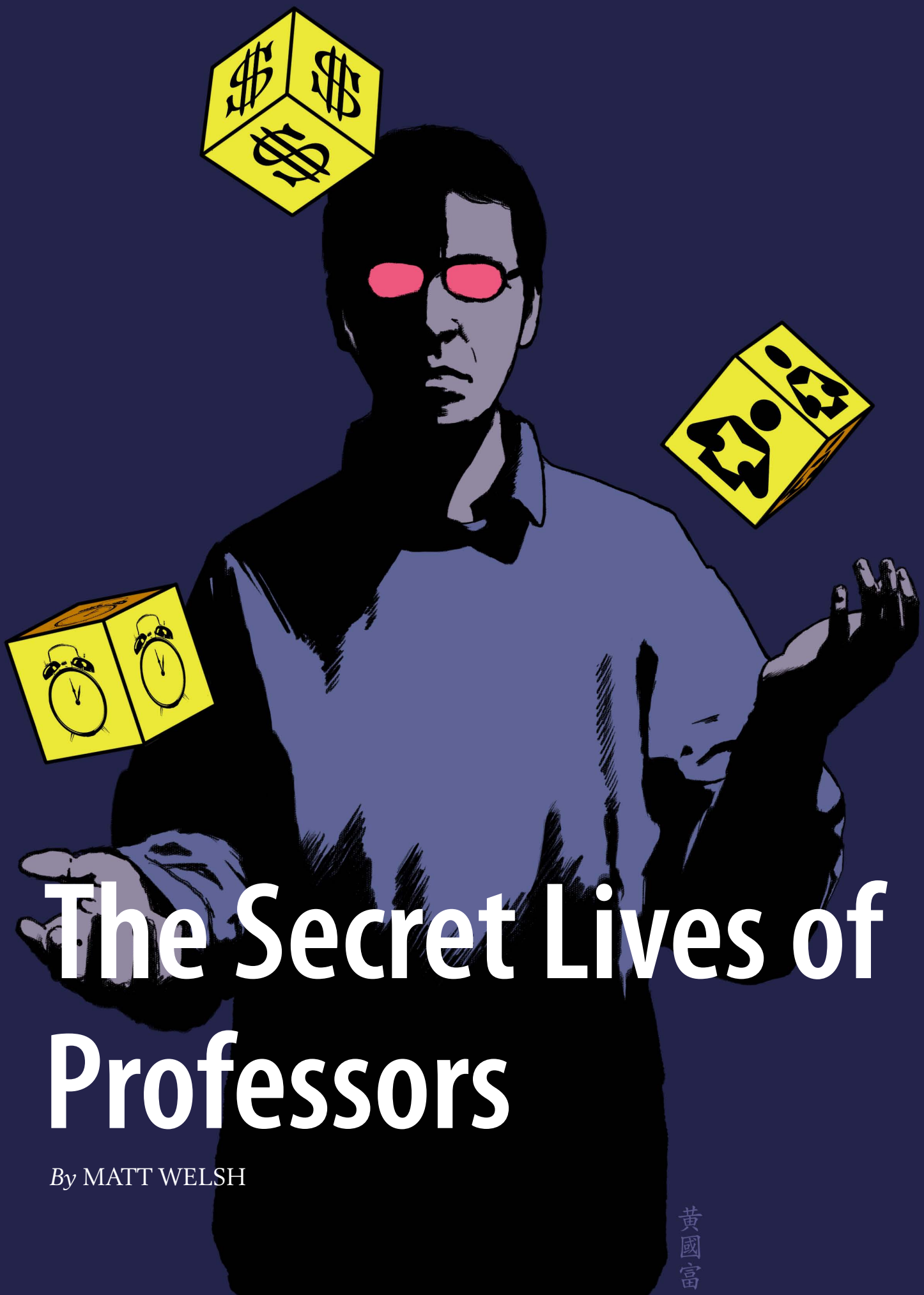
Scalable PHP Hosting, made easy with the CatN Platform.

Instantly deploy your apps on our super cluster. No code changes necessary. SSH Access, Cron jobs, Git & SVN support all as standard.

from £5 per month

Launching 1st July 2010

www.catn.com



The Secret Lives of Professors

By MATT WELSH

黃國富

I came to Harvard 7 years ago with a fairly romantic notion of what it meant to be a professor — I imagined unstructured days spent mentoring students over long cups of coffee, strolling through the verdant campus, writing code, pondering the infinite. I never really considered doing anything else. At Berkeley, the reigning belief was that the best and brightest students went on to be professors, and the rest went to industry — and I wanted to be one of those elite. Now that I have students that harbor their own rosy dreams of academic life, I thought it would be useful to reflect on what being a professor is really like. It is certainly not for everybody. It remains to be seen if it is even for me.

To be sure, there are some great things about this job. To first approximation you are your own boss, and even when it comes to teaching you typically have a tremendous amount of freedom. It has often been said that being a prof is like running your own startup — you have to hire the staff (the students), raise the money (grant proposals), and of course come up with the big ideas and execute on them. But you also have to do a lot of marketing (writing papers and giving talks), and sit on a gazillion stupid committees that eat up your time. This post is mostly for grad students who think they want to be profs one day. A few surprises and lessons from my time in the job...

Show me the money.

The biggest surprise is how much time I have to spend getting funding for my research. Although it varies a lot, I guess that I spent about 40% of my time chasing after funding, either directly (writing grant proposals) or indirectly (visiting companies, giving talks, building relationships). It is a huge investment of time that does not always contribute directly to your research agenda — just something you have to do to keep the wheels turning. To do systems research you need a lot of funding — at my peak I've had 8 Ph.D. students, 2 postdocs, and a small army of undergrads all working in my group. Here at Harvard, I don't have any colleagues working directly in my area, so I haven't been able to spread the fundraising load around very much. (Though huge props to Rob and Gu for getting us that $10M for RoboBees!) These days, funding rates are abysmal: less than 10% for some NSF programs, and the decision on a proposal is often arbitrary. And personally, I stink at writing proposals. I've had around 25 NSF proposals declined and only about 6 funded. My batting average for papers is much, much better. So, I can't let any potential source of funding slip past me.

Must... work... harder.

Another lesson is that a prof's job is never done. It's hard to ever call it a day and enjoy your "free time," since you can always be working on another paper, another proposal, sitting on another program committee, whatever. For years I would leave the office in the evening and sit down at my laptop to keep working as soon as I got home. I've heard a lot of advice on setting limits, but the biggest predictor of success as a junior faculty member is how much of your life you are willing to sacrifice. I have never worked harder than I have in the last 7 years. The sad thing is that so much of the work is for naught — I can't count how many hours I've sunk into meetings with companies that led nowhere, or writing proposals that never got funded. The idea that you get tenure and sit back and relax is not quite accurate — most of the tenured faculty I know here work even harder than I do, and they spend more of their time on stuff that has little to do with research.

Your time is not your own.

Most of my days are spent in an endless string of meetings. I find almost no time to do any hacking anymore, which is sad considering this is why I became a computer scientist. When I do have some free time in my office it is often spent catching up on email, paper reviews, random paperwork that piles up when you're not looking. I have to delegate all the fun and interesting problems to my students. They don't know how good they have it!

Students are the coin of the realm.

David Patterson once said this and I now know it to be true. The main reason to be an academic is not to crank out papers or to raise a ton of money but to train the next generation. I love working with students and this is absolutely the best part of my job. Getting in front of a classroom of 80 students and explaining how virtual memory works never ceases to be thrilling. I have tried to mentor my grad students, though in reality I have learned more from them than they will ever learn from me. My favorite thing is getting undergrads involved in research, which is how I got started on this path as a sophomore at Cornell, when Dan Huttenlocher took a chance on this long-haired crazy kid who skipped his class a lot. So I try to give back.

Of course, my approach to being a prof is probably not typical. I know faculty who spend a lot more time in the lab and a lot less time doing management than I do. So there are lots of ways to approach the job — but it certainly was not what I expected when I came out of grad school. ■

Matt Welsh is a professor of Computer Science at Harvard University. His research interests include OS, network, and programming language support for complex, large-scale systems, including wireless sensor networks and cloud computing services. He is the author of "Running Linux" and blogs at *http://matt-welsh.blogspot.com*.

Reprinted with permission of the original author. First appeared in *http://matt-welsh.blogspot.com/2010/05/secret-lives-of-professors.html.*

New Programming Jargon

By JOEY DEVILLA

Every field comes up with its own jargon, and oftentimes subgroups within a field come up with their own specific words or phrases (those of you familiar with Microsoft Canada's Developer and Platform Evangelism Team know that we have our own term for "broken", named after one of our teammates who is notorious for killing all sorts of tech gear).

A question recently posted on Stack Overflow asked for people to submit programming terms that they or their team have coined and have come into regular use in their own circles. I took a number of the submissions and compiled them into the alphabetically ordered list below for your education and entertainment.

Banana Banana Banana

Placeholder text indicating that documentation is in progress or yet to be completed. Mostly used because FxCop complains when a public function lacks documentation.

Example:

```
/// <summary>
/// banana banana banana
/// </summary>
public CustomerValidationResponse
Validate(CustomerValidationRequ
est request, bool ...
```

Barack Obama

A project management account to which the most aspirational tickets – stuff you'd really like to do but will pobably never get approval for – gets assigned.

Bicrement

Adding 2 to a variable.

Bloombug

A bug that accidentally generates money. [Joey's note: I have never written one of these.]

Bugfoot

A bug that isn't reproducible and has been sighted by only one person. See *Loch Ness Monster Bug*.

Chunky Salsa

A single critical error or bug that renders an entire system unusable, especially in a production environment.

Based on the chunky salsa rule from *TVTropes*: Any situation that would reduce a character's head to the consistency of chunky salsa dip is fatal, regardless of other rules.

Configuration Programming / Programmer

Someone that says they are a programmer but only knows how to hack at configuration files of some other pieces of software configuration to make them do what they want.

Counterbug

A defensive move useful for code reviews. If someone reviewing your code presents you with a bug that's your fault, you counter with a counterbug: a bug caused by the reviewer.

DOCTYPE Decoration

When web designers add a proper DOCTYPE declaration at the beginning of an HTML document, but then don't bother to write valid markup for the rest of it.

Drug Report

A bug report so utterly incomprehensible that whoever submitted it must have been smoking crack. The lesser version is a chug report, where the submitter is thought have had one too many.

Duck

A feature added for no other reason than to draw management attention and be removed, thus avoiding unnecessary changes in other aspects of the product.

This started as a piece of Interplay corporate lore. It was well known that producers (a game industry position, roughly equivalent to PMs) had to make a change to everything that was done. The assumption was that subconsciously they felt that if they didn't, they weren't adding value.

The artist working on the queen animations for Battle Chess was aware of this tendency, and came up with an innovative solution. He did the animations for the queen the way that he felt would be best, with one addition: he gave the queen a pet duck. He animated this duck through all of the queen's animations, had it flapping around the corners. He also took great care to make sure that it never overlapped the "actual" animation.

Eventually, it came time for the producer to review the animation set for the queen. The producer sat down and watched all of the animations. When they were done, he turned to the artist and said, "That looks great. Just one thing – get rid of the duck."

Fear-Driven Development

When project management adds more pressure, such as by firing a member of the team.

Ghetto Code

A particularly inelegant and obviously suboptimal section of code that still meets the requirements. [Joey's note: I've written ghetto code before, but that's because I'm street, yo!]

Hindenbug

A catastrophic data-destroying bug. Oh, the humanity!

Hocus Focus Problem

Unexpected behavior caused by changes in focus, or incorrect setting of focus. Could also be used to describe an app stealing your focus.

Hot Potato / Hot Potatoes

A fun way to pronounce http:// and https://.

IRQed

Annoyed by interruptions. Pronounced like and has a similar meaning to "irked".

Jimmy

A generalized name for the clueless/new developer. The submitter at Stack Overflow writes:

> *We found as we were developing a framework component that required minimal knowledge of how it worked for the other developers. We would always phrase our questions as: "What if Jimmy forgets to update the attribute?"*
> *This led to the term "Jimmy-proof" when referring to well designed framework code.*

It's probably best not to use this term around IronRuby developer Jimmy Schementi.

Loch Ness Monster Bug

A bug that isn't reproducible and has been sighted by only one person. See *Bugfoot*.

Megamoth

MEGA MOnolithic meTHod. Usually stretches over two screens in height and often contained inside a God Object (an object that knows or does too much).

.NET Sandwich

When .NET code called native code which calls other .NET code and makes the poorly designed application crash.

n-gleton
A class that only allows a fixed number of instances of itself.

NOPping
Not napping, but simply zoning out. Comes from the assembly language instruction NOP, for No OPeration, which does nothing.

Pokemon Exception Handling
For when you just gotta catch 'em all!

Reality 101 Failure
The program (or more likely feature of a program) does exactly what was asked for, but when it's deployed it turns out that the problem was misunderstood and the program is basically useless.

Refuctoring
The process of taking a well-designed piece of code and, through a series of small, reversible changes, making it completely unmaintainable by anyone except yourself. It's job security!

The Sheath
The isolating interface between your team's (good) code, and the brain-dead code contributed by some other group. The sheath prevents horrible things (badly named constants, incorrect types, etc.) in their code from infecting your code.

Shrug Report
A bug report with no error message or "how to reproduce" steps and only a vague description of the problem. Usually contains the phrase "doesn't work."

Smug Report
A bug report submitted by a user who thinks he knows a lot more about the system's design than he really does. Filled with irrelevant technical details and one or more suggestions (always wrong) about what he thinks is causing the problem and how we should fix it.

Stringly-Typed
A riff on *strongly-typed*. Used to describe an implementation that needlessly relies on strings when programmer- and refactor-friendly options are available.

Examples:

- Method parameters that take strings when other more appropriate types should be used
- On the occasion that a string is required in a method call (e.g. network service), the string is then passed and used throughout the rest of the call graph without first converting it to a more suitable internal representation (e.g. parse it and create an enum, then you have strong typing throughout the rest of your codebase)
- Message passing without using typed messages etc.

Excessively stringly typed code is usually a pain to understand and detonates at runtime with errors that the compiler would normally find.

Unicorny
An adjective to describe a feature that's so early in the planning stages that it might as well be imaginary. This one comes from Rails Core Team member Yehuda Katz, who used it in his closing keynote at last year's Windy City Rails to describe some of Rails' upcoming features.

```
if (5 == count)
```

Yoda Conditions
The act of using

```
if (constant == variable)
```

instead of

```
if (variable == constant)
```

It's like saying "If blue is the sky".■

Joey deVilla is Microsoft Canada's unlikely Developer Evangelist. Prior to working for "The Empire", he worked on open source software at a number of startups, developed multimedia CD-ROMs, worked the street as an accordion busker and even had a stint as an accordion-playing go-go dancer at a Toronto nightclub. You'll often find him hanging out at Toronto's hackerspace HacklabTO.

Reprinted with permission of the original author. First appeared in *http://www.globalnerdy.com/2010/05/09/new-programming-jargon/*.

Scooping the Loop Snooper

A proof that the Halting Problem is undecidable

By GEOFFREY K. PULLUM

No general procedure for bug checks succeeds.
Now, I won't just assert that, I'll show where it leads:
I will prove that although you might work till you drop,
you cannot tell if computation will stop.

For imagine we have a procedure called P
that for specified input permits you to see
whether specified source code, with all of its faults,
defines a routine that eventually halts.

You feed in your program, with suitable data,
and P gets to work, and a little while later
(in finite compute time) correctly infers
whether infinite looping behavior occurs.

If there will be no looping, then P prints out 'Good.'
That means work on this input will halt, as it should.
But if it detects an unstoppable loop,
then P reports 'Bad!' — which means you're in the soup.

Well, the truth is that P cannot possibly be,
because if you wrote it and gave it to me,
I could use it to set up a logical bind
that would shatter your reason and scramble your mind.

Here's the trick that I'll use — and it's simple to do.
I'll define a procedure, which I will call Q,
that will use P's predictions of halting success
to stir up a terrible logical mess.

For a specified program, say A, one supplies,
the first step of this program called Q I devise
is to find out from P what's the right thing to say
of the looping behavior of A run on A.

If P's answer is 'Bad!', Q will suddenly stop.
But otherwise, Q will go back to the top,
and start off again, looping endlessly back,
till the universe dies and turns frozen and black.

And this program called Q wouldn't stay on the shelf;
I would ask it to forecast its run on itself.
When it reads its own source code, just what will it do?
What's the looping behavior of Q run on Q?

If P warns of infinite loops, Q will quit;
yet P is supposed to speak truly of it!
And if Q's going to quit, then P should say 'Good.'
Which makes Q start to loop! (P denied that it would.)

No matter how P might perform, Q will scoop it:
Q uses P's output to make P look stupid.
Whatever P says, it cannot predict Q:
P is right when it's wrong, and is false when it's true!

I've created a paradox, neat as can be —
and simply by using your putative P.
When you posited P you stepped into a snare;
Your assumption has led you right into my lair.

So where can this argument possibly go?
I don't have to tell you; I'm sure you must know.
By reductio, there cannot possibly be
a procedure that acts like the mythical P.

You can never find general mechanical means
for predicting the acts of computing machines.
It's something that cannot be done. So we users
must find our own bugs. Our computers are losers!

Geoffrey K. Pullum is a linguist, currently teaching at the University of Edinburgh. Formerly he was at the University of California, Santa Cruz. His main research interests for some time have been in the grammar of Standard English and the formalization of syntactic theories, and his recreational interest in theoretical computer science arises out of the latter.

Reprinted with permission of the original author. First appeared in *http://ling.ed.ac.uk/~gpullum/loopsnoop.html.* An earlier version was published in Mathematics Magazine (73).

Programming Books
The C Programming

By MIKE TAYLOR

It's 32 YEARS old, and it remains the single greatest book ever written about a programming language. Its crown is secure; even if you'd not already read the title of this article, you'd know what book I'm talking about. It's the only language-specific book in Top Five programming books of the Programming Reddit's FAQ. Co-written by Reinvigorated Programmer regular Brian W. Kernighan and Dennis M. Ritchie, it's not just the definitive book about the language in question, it's the book that rewrote the book on what it means to be definitive. Step forward, please, The C Programming Language!

The biography of the Beatles at allmusic.com has a very astute and resonant bit of analysis right in the first paragraph, saying that "they were among the few artists of any discipline that were simultaneously the best at what they did and the most popular at what they did." You could say the same for K&R, as it's affectionately known: everyone knows it's the best book on C, and (for once) the thing that everyone knows is actually true.

So what makes it so great?

Short, comprehensive, dense

First: it's so short. At 272 pages (this is for the second edition, published in 1988 and describing ANSI C), it's shorter that Harry Potter and the Prisoner of Azkaban (317 pages) and little more than one third the length of Order of the Phoenix.

Second, it's so comprehensive. There is, essentially, nothing to be known about C beyond what is in this book. If you can read those 272 pages, and understand them all, then you are well on the way to being a C wizard. (Er, assuming you have the patience to go on to accumulate a decade of experience leading to wisdom, taste, good judgement and technical intuition.)

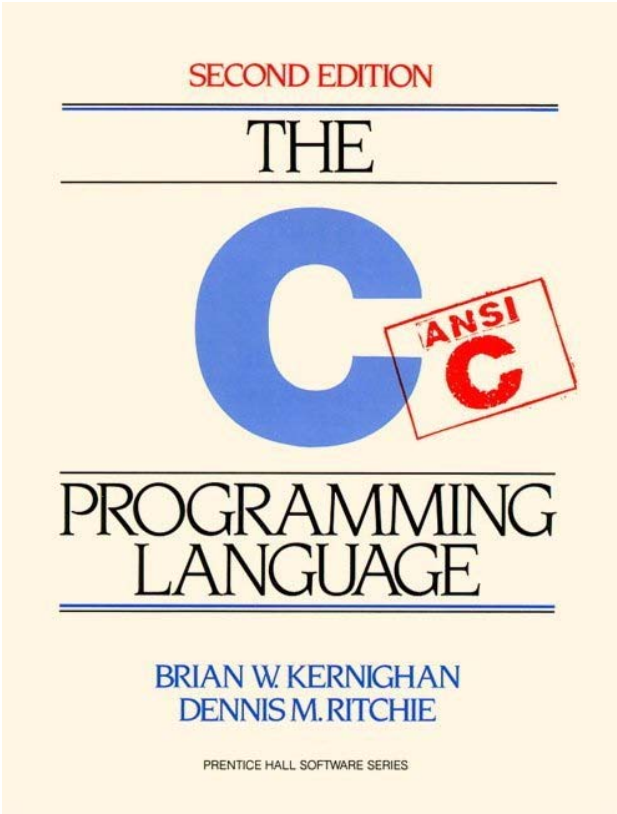
Third, and this is really a consequence of the first two, it's so dense. This is not a book that wastes words. There are no extended introductory sections on Why You Should Learn C and C's Place In The World. The two prefaces (for 1st and 2nd editions) are one and a bit pages each. The introduction is four pages. Then we're straight into Chapter 1. A Tutorial Introduction, which is 30 pages long and in that space covers:

- 1.1 Getting Started
- 1.2 Variables and Arithmetic Expressions
- 1.3 The For Statement
- 1.4 Symbolic Constants
- 1.5 Character Input and Output
- 1.6 Arrays
- 1.7 Functions
- 1.8 Arguments — Call by Value
- 1.9 Character Arrays
- 1.10 External Variables and Scope

At the end of that chapter, on page 34, is a sequence of five exercises, culminating in this one (and enjoy the characteristic Kernighanian understatement in the final sentence):

> ***Exercise 1-24***. *Write a program to check a C program for rudimentary syntax errors like unbalanced parentheses, brackets and braces. Don't forget about quotes, both single*

: Language



> *and double, escape sequences, and comments. (This program is hard if you do it in full generality.)*

And, as tough as that may seem after only 30 pages, they really have given you all the tools you need to do the exercise by this point.

Say what you mean, simply and directly

Apologies if you're getting bored of reading this Kernighan-and-Plauger epigram every time you return to this blog, but I really don't think it can be over-emphasised. Although this advice's appearance in The Elements of Programming Style is of course in the context of writing programs, Kernighan also follows his own advice when it comes to writing prose. No words are wasted; neither is your time. Yet somehow the book avoids feeling rushed despite packing so much into so little space.

After the tutorial introduction, the remaining chapters cover:

- Chapter 2. Types, Operators, and Expressions
- Chapter 3. Control Flow
- Chapter 4. Functions and Program Structure
- Chapter 5. Pointers and Arrays [this, by the way, on page 93]
- Chapter 6. Structures
- Chapter 7. Input and Output
- Chapter 8. The UNIX System Interface

That's it for the chapters. So they've got you doing systems programming by page 169; from page 185 to the end of the chapter, they show you how to implement `malloc()`. These guys are not messing about.

And then it's on to the appendices, which rival those of The Return of the King for comprehensiveness (though thankfully without the notes on the differences between Eldar and Númenorean calendars).

- Appendix A. Reference Manual [because all the chapters are tutorial]
- Appendix B. Standard Library [yes, all of it, in 18 pages]
- Appendix C. Summary of Changes [since the 1st edition]

And finally, there's just time for a characteristically comprehensive index before the book comes to a close.

In praise of small

Kernighan and Ritchie's much-quoted preface explains the philosophy behind the book's characteristically dense structure:

> *We have tried to retain the brevity of the first edition. C is not a big language, and it is not well served by a big book. [...] Appendix A, the reference manual, is not the standard, but our attempt to convey the essentials of the standard in a smaller space. [...] As we said in the preface to the first edition, C "wears well as one's experience with it grows." With a decade more experience, we still feel that way.*

And it's true that the book is only able to be as short as it is because the language that it describes is as small as it is. I have the second edition of Stroustrup's The C++ Programming Language, which clearly models itself on K&R and is about as terse as such

Kernighan, left, railing against innumeracy; Ritchie, right, auditioning for the role of Saruman.

a book can be, but its 691 pages make it fully two and half times the size of the original. This, mind you, is the second edition of Stroustrup, published in 1991 only three years after the K&R second edition, when C++ was still relatively well under control.

There is much, much more that I could say about the smallness of C, but rather that go against everything I've just been saying by bloating this review up into a monster, I am going to save that for a separate article.

Do it yourself

It's also characteristic of K&R that they have this statement on the copyright page:

> *This book was typeset (**pic**|**tbl**|**eqn**|**troff -ms**) in Times Roman and Courier by the authors, using an Autologic APS-5 phototypesetter and a DEC VAX 8550 running the 9th Edition of the UNIX(R) operating system.*

That they did their own typesetting is not just a cute touch, but an insight on the completeness of their mastery of what they were doing, and the care they took over it. The book is not what you would call beautiful to look at, but the typesetting is wholly functional, at one with the text rather than fighting against it.

If I could analyse it, I'd do it myself

Finally, we come to the aspect of The C Programming Language that is hardest to explain — and hardest to do.

The bottom line here is that writing is an art. You can hack your way through to producing tolerable text without being an artist, just as an uninspired programmer can bash his way through to wiring together an uninspired web application. But just as it takes a Ken Thompson to invent and write UNIX, and a Dennis Ritchie to invent C and write the initial compiler, so it takes a Brian Kernighan to write The C Programming Language.

If all it took to write a truly great technical book was to write down everything there is to say about a subject and then ruthlessly distill it to its essence, then great technical books would be much less rare than they are. That, I think, is a prerequsite; but it's Necessary But Not Sufficient. There is a graceful quality about the writing in K&R, even when it is brutally technical; it draws you on and in; it's just pleasant to read. It is, on occasion, gently humorous, though certainly not written for laughs the way that, say, Programming Perl is. It's exhilarating how the book takes you somewhere worth getting to, and does it so quickly. It treats you like a grown-up; it is not "For Dummies", but its intelligent approach is not the elitist kind that seems to want to make the reader feel inferior, but a warm intelligence that lifts you up to its level. In short, it's a book that wants to make you a better programmer.

The best way I can express it is to say that at the end of each section and subsection, you want to read on and find out what's next. That stands in stark contrast to too many other technical books, where I find myself peeking ahead to find out how much more of the current chapter there is to plough through before I can stop reading.

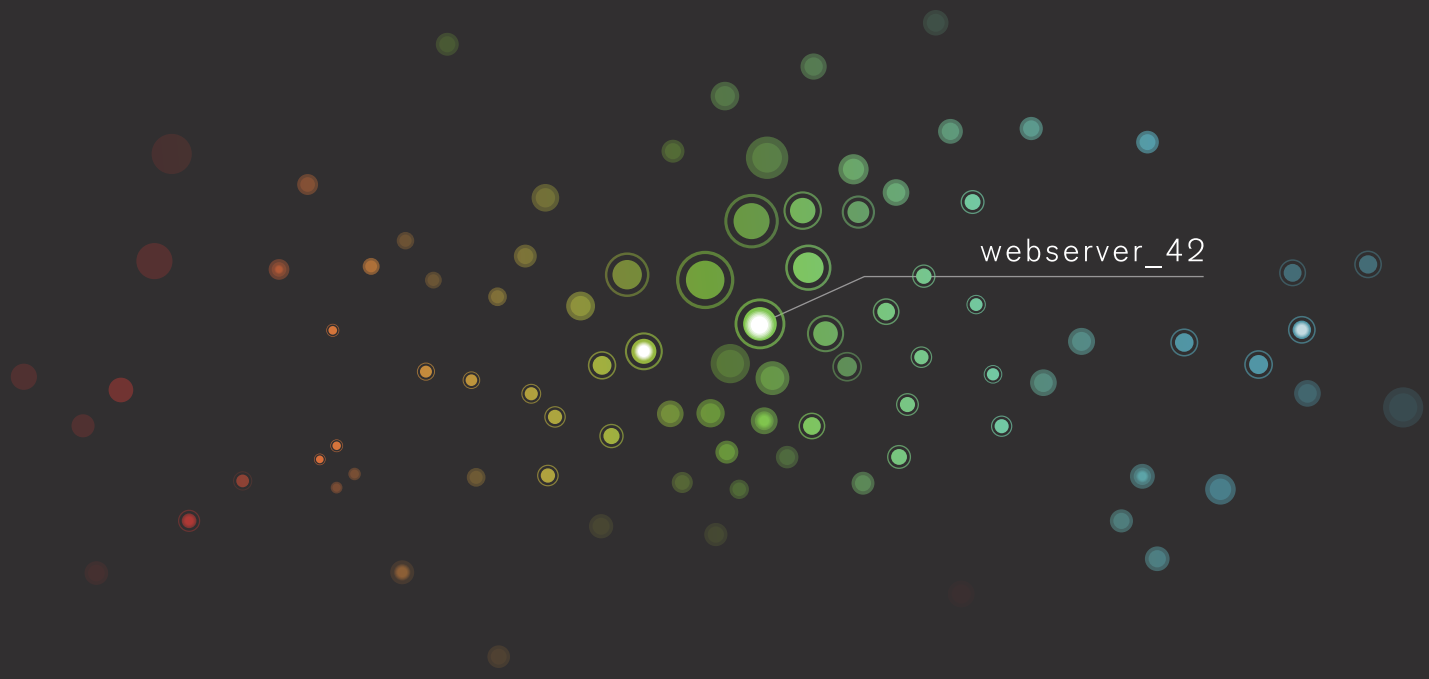
I wish I knew how they did it. But I'm glad that they did. Kernighan and Ritchie, we salute you! ■

Mike Taylor is a computer programmer by day and a dinosaur palaeontologist by night, twin obsessions reflected in his two blogs, http://reprog.wordpress.com/ and http://svpow.wordpress.com/. He started programming in 1980, on a Commodore PET 2001 and a Video Genie, and has hardly stopped since.

Reprinted with permission of the original author. First appeared in *http://reprog.wordpress.com/2010/04/06/programming-books-part-4-the-c-programming-language/*.

These are your servers

These are your servers on Cloudkick



Any questions?

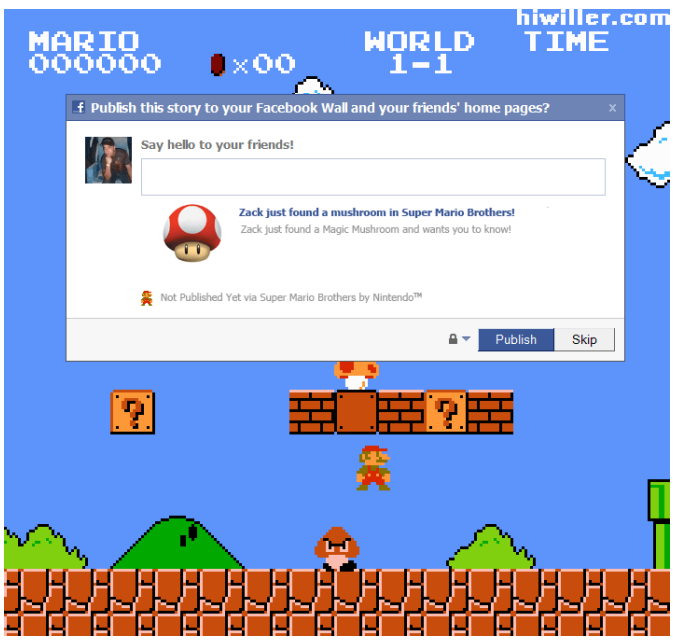
cloudkick.com
415.779.5425

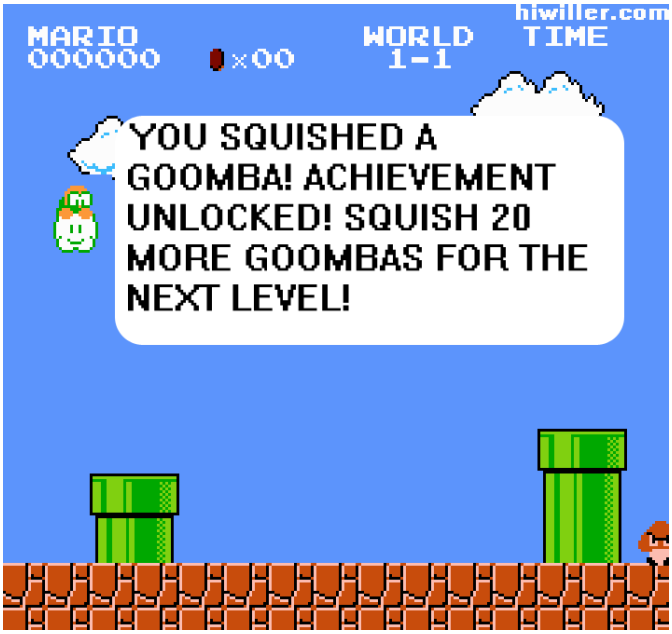
support for 8 clouds + dedicated hardware

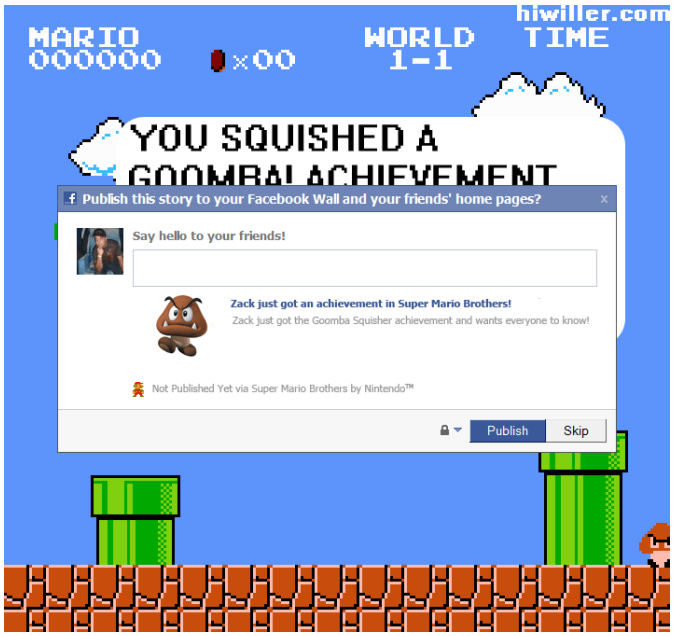
cloudkick
the best way to manage the cloud

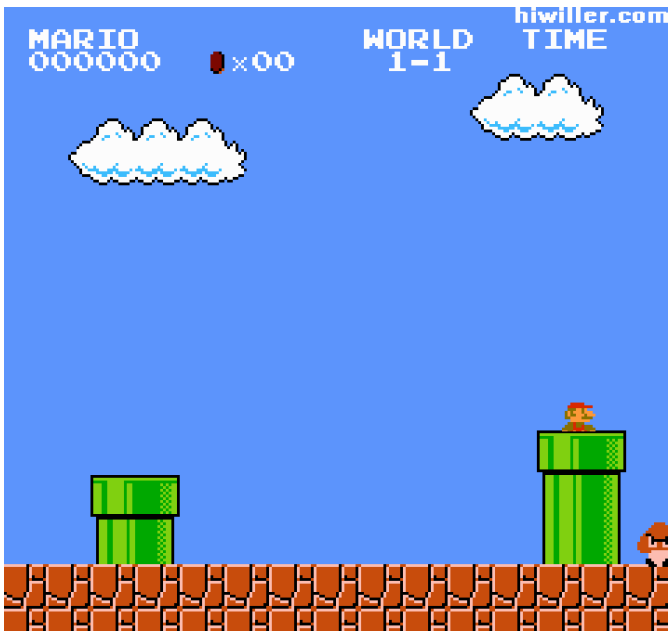
If Mario Was Designed in 2010

By ZACK HIWILLER









Author's Note

While this post is meant to be humorous, it isn't meant to be humorous at the expense of my fellow designers. I know we all try to do what is best for our games and Lord knows I am just as guilty as everyone else, so don't take offense guys! It's just me pining for a simpler time…■

Zack Hiwiller is a game designer currently living in New York City. He's worked on games on eleven platforms from the lowly Game Boy Advance to the chugging heat-expelling behemoth called the Playstation 3. He writes about games and the game industry on his blog at http://www.hiwiller.com.

Reprinted with permission of the original author. First appeared in *http://www.hiwiller.com/2010/04/29/if-mario-was-designed-in-2010/*.

Worst-Case Thinking

BY BRUCE SCHNEIER

At a security conference recently, the moderator asked the panel of distinguished cybersecurity leaders what their nightmare scenario was. The answers were the predictable array of large-scale attacks: against our communications infrastructure, against the power grid, against the financial system, in combination with a physical attack.

I didn't get to give my answer until the afternoon, which was: "My nightmare scenario is that people keep talking about their nightmare scenarios."

There's a certain blindness that comes from worst-case thinking. An extension of the precautionary principle, it involves imagining the worst possible outcome and then acting as if it were a certainty. It substitutes imagination for thinking, speculation for risk analysis, and fear for reason. It fosters powerlessness and vulnerability and magnifies social paralysis. And it makes us more vulnerable to the effects of terrorism.

Worst-case thinking means generally bad decision making for several reasons. First, it's only half of the cost-benefit equation. Every decision has costs and benefits, risks and rewards. By speculating about what can possibly go wrong, and then acting as if that is likely to happen, worst-case thinking focuses only on the extreme but improbable risks and does a poor job at assessing outcomes.

Second, it's based on flawed logic. It begs the question by assuming that a proponent of an action must prove that the nightmare scenario is impossible.

Third, it can be used to support any position or its opposite. If we build a nuclear power plant, it could melt down. If we don't build it, we will run short of power and society will collapse into anarchy. If we allow flights near Iceland's volcanic ash, planes will crash and people will die. If we don't, organs won't arrive in time for transplant operations and people will die. If we don't invade Iraq, Saddam Hussein might use the nuclear weapons he might have. If we do, we might destabilize the Middle East, leading to widespread violence and death.

Of course, not all fears are equal. Those that we tend to exaggerate are more easily justified by worst-case thinking. So terrorism fears trump privacy fears, and almost everything else; technology is hard to understand and therefore scary; nuclear weapons are worse than conventional weapons; our children need to be protected at all costs; and annihilating the planet is bad. Basically, any fear that would make a good movie plot is amenable to worst-case thinking.

Fourth and finally, worst-case thinking validates ignorance. Instead of focusing on what we know, it focuses on what we don't know — and what we can imagine.

Remember Defense Secretary Rumsfeld's quote? "Reports that say that something hasn't happened are always interesting to me, because as we know, there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns — the ones we don't know we don't know." And this: "the absence of evidence is not evidence

of absence." Ignorance isn't a cause for doubt; when you can fill that ignorance with imagination, it can be a call to action.

Even worse, it can lead to hasty and dangerous acts. You can't wait for a smoking gun, so you act as if the gun is about to go off. Rather than making us safer, worst-case thinking has the potential to cause dangerous escalation.

The new undercurrent in this is that our society no longer has the ability to calculate probabilities. Risk assessment is devalued. Probabilistic thinking is repudiated in favor of "possibilistic thinking": Since we can't know what's likely to go wrong, let's speculate about what can possibly go wrong.

Worst-case thinking leads to bad decisions, bad systems design, and bad security. And we all have direct experience with its effects: airline security and the TSA, which we make fun of when we're not appalled that they're harassing 93-year-old women or keeping first graders off airplanes. You can't be too careful!

Actually, you can. You can refuse to fly because of the possibility of plane crashes. You can lock your children in the house because of the possibility of child predators. You can eschew all contact with people because of the possibility of hurt. Steven Hawking wants to avoid trying to communicate with aliens because they might be hostile; does he want to turn off all the planet's television broadcasts because they're radiating into space? It isn't hard to parody worst-case thinking, and at its extreme it's a psychological condition.

Frank Furedi, a sociology professor at the University of Kent, writes: "Worst-case thinking encourages society to adopt fear as one of the dominant principles around which the public, the government and institutions should organize their life. It institutionalizes insecurity and fosters a mood of confusion and powerlessness. Through popularizing the belief that worst cases are normal, it incites people to feel defenseless and vulnerable to a wide range of future threats."

Even worse, it plays directly into the hands of terrorists, creating a population that is easily terrorized — even by failed terrorist attacks like the Christmas Day underwear bomber and the Times Square SUV bomber.

When someone is proposing a change, the onus should be on them to justify it over the status quo. But worst-case thinking is a way of looking at the world that exaggerates the rare and unusual and gives the rare much more credence than it deserves.

It isn't really a principle; it's a cheap trick to justify what you already believe. It lets lazy or biased people make what seem to be cogent arguments without understanding the whole issue. And when people don't need to refute counterarguments, there's no point in listening to them. ■

Internationally renowned security expert Bruce Schneier has authored nine books — including Schneier on Security and Beyond Fear — and hundreds of articles and academic papers. Schneier regularly appears on television and radio, has testified before Congress, and is a frequent writer and lecturer on issues surrounding security and privacy.

Reprinted with permission of the original author. First appeared in *http://www.schneier.com/blog/archives/2010/05/worst-case_thin.html.*

9 Years of Sleep

By DOMINIC SZABLEWSKI

For the last ten years or so, I used to turn on my PC when I came home from school or work and shut it down again right before I went to bed. So most of the time when my PC is running, I'm awake. I've also been idling in IRC for as long as I had Internet – when my PC is running, so is my IRC client.

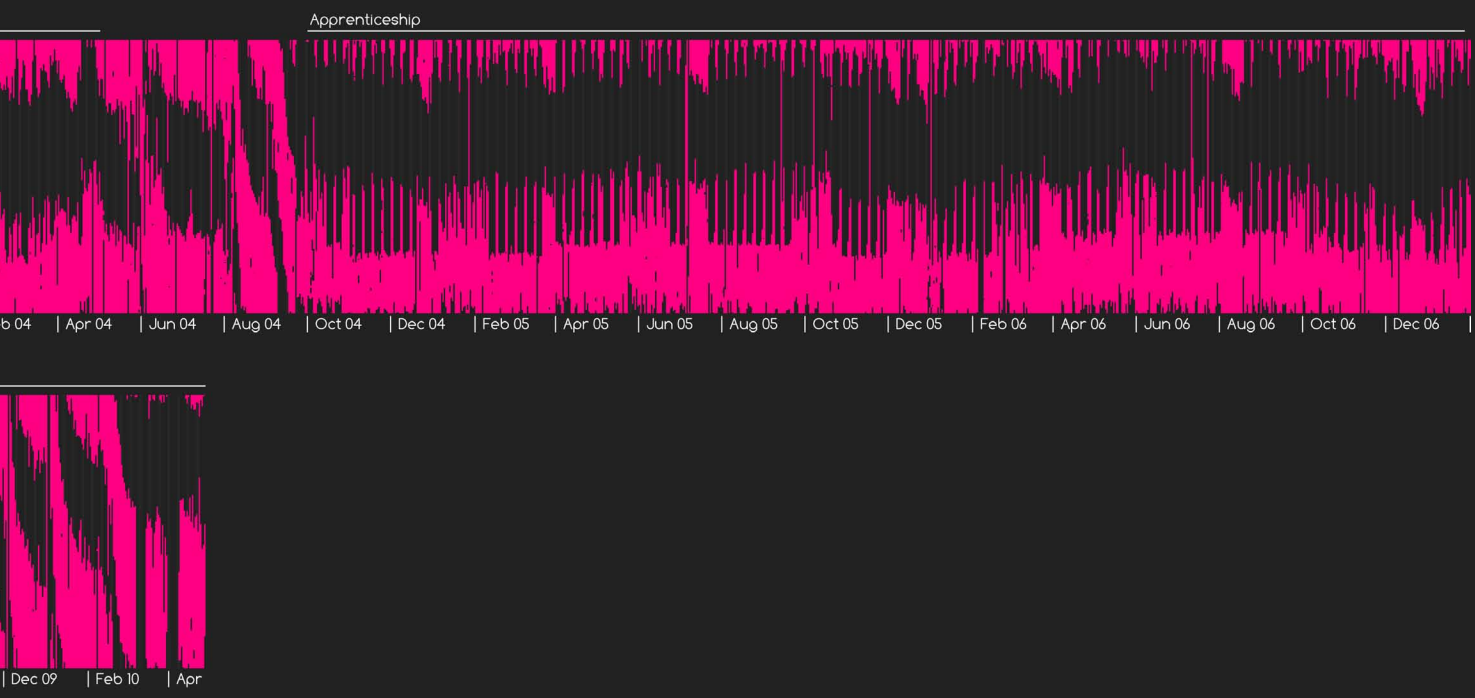
I still have all my IRC logs since 2001 lying on my HDD. The log format of mIRC changed slightly over the years, but it's all easily parsable with some basic Regexp. I quickly wrote a PHP script that extracts the Session Start and Sessions Close markers and timestamps from these logs and transfers them into an image.

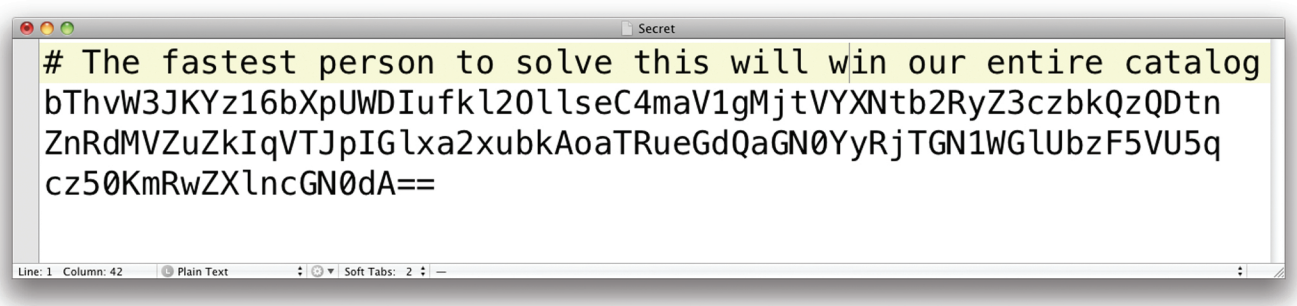
As you can see, I tend to stay up late. I also tend to go into a free-running sleep mode when I don't have to get up early every morning. During May 2004, after my A-Level exams and before my apprenticeship started, I "rotated" my sleep cycle three times. This has been even more extreme in the last two years, when we've had fewer lectures and instead worked on a lot of projects. I should really get one of these daylight lamps.

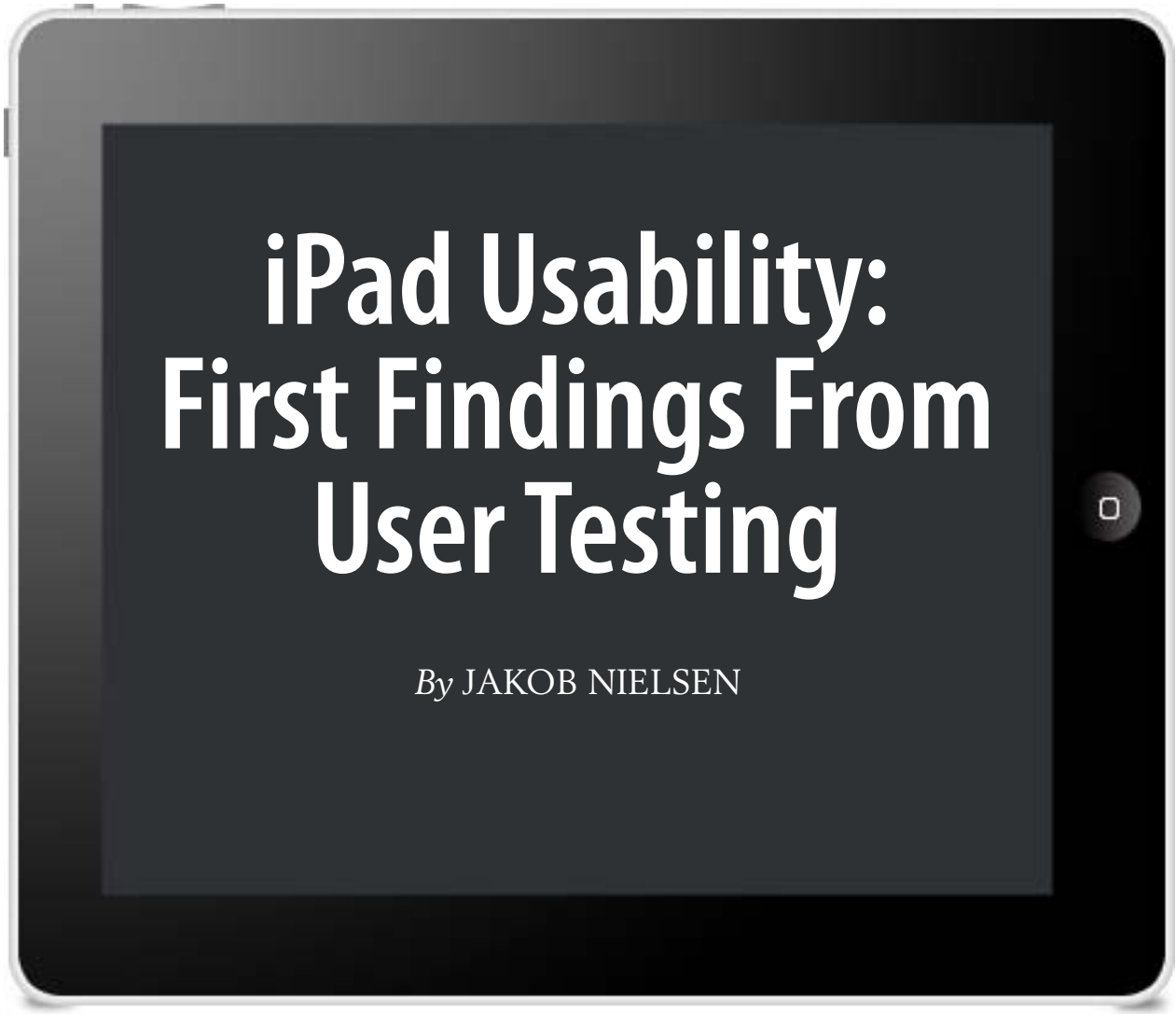
There's so much more interesting information hidden in these IRC logs. Maybe I can bring myself to parse and import all of them into a database, so I can run some simple queries on them. Maybe I can even find my pre-2001 IRC logs on some backup CDs. ■

Dominic Szablewski is a freelance developer and a student for Digital Media at the Hochschule Darmstadt in Germany. He is currently working on his bachelor thesis about real-time games written using HTML5. PhobosLab is his personal blog about any project he can get to a presentable state.

Reprinted with permission of the original author. First appeared in *http://www.phoboslab.org/log/2010/05/9-years-of-sleep.*







iPad Usability: First Findings From User Testing

By JAKOB NIELSEN

IT LOOKS LIKE a giant iPhone", is the first thing users say when asked to test an iPad. (Their second comment? "Wow, it's heavy.")

But from an interaction design perspective, an iPad user interface shouldn't be a scaled-up iPhone UI.

Indeed, one finding from our study is that the tab bar at the bottom of the screen works much worse on iPad than on iPhone. On the small phone, users are likely to notice the muted icons at the bottom of the screen, even if their attention is on content in the middle of the screen. But the iPad's much bigger screen means that users are typically directing their gaze far from the tab bar and they ignore (and forget) those buttons.

Another big difference between iPad and iPhone is that regular websites work reasonably well on the big tablet. In our iPhone usability studies, users strongly prefer using apps to going on the Web. It's simply too painful to use most websites on the small screen. (Mobile-optimized sites alleviate this issue, but even they usually have worse usability than apps.)

The iPad's bigger screen offers reasonable usability for regular Web pages. Of course, there's still the "fat finger" problem common to all touch screens, which makes it hard for users to reliably hit small targets. The iPad has a read–tap asymmetry, where text big enough to read is too small to touch. Thus, we definitely recommend large touch zones on any Web page hoping to attract many iPad users.

Also, most Web pages offer a rich and overstuffed experience compared to the iPad's sparse and regulated environment; when an iPad app suddenly launches users onto the Web, the transition can be jarring.

For more than a decade, when we ask users for their first impression of (desktop) websites, the most

frequently-used word has been "busy." In contrast, the first impression of many iPad apps is "beautiful." The change to a more soothing user experience is certainly welcome, especially for a device that may turn out to be more of a leisure computer than a business computer. Still, beauty shouldn't come at the cost of being able to actually use the apps to derive real benefits from their features and content.

First Studies

We conducted our initial usability studies of iPad apps and content a few weeks after Apple launched the device. We tested 7 users — all with at least 3 months' iPhone experience — but only one was an "experienced" iPad user.

(This user had only a week's experience — far less than the minimum of one year's experience that we usually request of usability study participants.)

Obviously, the findings from this research are only preliminary. However, we're releasing them anyway because the iPad platform is so different and is expected to attract considerable application development during the coming months. It would be a shame for all these apps to be designed without the benefit of the usability insights that do exist, despite the gaps in our current knowledge.

We tested the following applications and websites:

- ABC player
- Alice in Wonderland Lite
- AP News
- Art Authority
- BBC News
- Bloomberg
- craigsphone (Craigslist)
- eBay (both app and website)
- The Elements (physics courseware)
- Endless.com
- Epicurious
- ESPN Score Center
- ESPN.com
- Gap
- Gilt
- GQ magazine
- GWR Lite (Guinness World Records)
- iBook
- IMDb (Internet Movie Database)
- iverse Comics
- Kayak (kayak.com)
- Marvel Comics
- MLB.com (Major League Baseball)
- Nike.com
- Now Playing
- NPR (National Public Radio)
- The New York Times Editors' Choice
- Popular Science
- Time Magazine
- USA Today
- virginamerica.com
- whitehouse.gov
- Wolfram Alpha
- Yahoo! Entertainment

> "Anything you can show and touch can be a UI on this device."

Wacky Interfaces

The first crop of iPad apps revived memories of Web designs from 1993, when Mosaic first introduced the image map that made it possible for any part of any picture to become a UI element. As a result, graphic designers went wild: anything they could draw could be a UI, whether it made sense or not.

It's the same with iPad apps: anything you can show and touch can be a UI on this device. There are no standards and no expectations.

Worse, there are often no perceived affordances for how various screen elements respond when touched. The prevailing aesthetic is very much that of flat images that fill the screen as if they were etched. There's no lighting model or pseudo-dimensionality to indicate raised or lowered visual elements that call out to be activated.

In contrast, long-standing GUI design guidelines for desktop user designs dictate that buttons look raised (and thus pressable) and that scrollbars and other interactive elements are visually distinct from the content.

The traditional GUI separation between "church and state" — that is, between content and features or commands — has carried over to modern Web design. Those 1993-style image maps are long gone from any site that hopes to do business on the Internet.

The iPad etched-screen aesthetic does look good. No visual distractions or nerdy buttons. The penalty for this beauty is the re-emergence of a usability problem we haven't seen since the mid-1990s: Users don't know where they can click.

For the last 15 years of Web usability research, the main problems have been that users don't know where to go or which option to choose — not that they don't even know which options exist. With iPad UIs, we're back to this square one.

Inconsistent Interaction Design

To exacerbate the problem, once they do figure out how something works, users can't transfer their skills from one app to the next. Each application has a completely different UI for similar features.

In different apps, touching a picture could produce any of the following 5 results:

> **"A strategic issue for iPad user experience design is whether to emphasize user empowerment or author authority."**

- Nothing happens
- Enlarging the picture
- Hyperlinking to a more detailed page about that item
- Flipping the image to reveal additional pictures in the same place (metaphorically, these new pictures are "on the back side" of the original picture)
- Popping up a set of navigation choices

The latter design was used by USA Today: Touching the newspaper's logo brought up a navigation menu listing the various sections. This was probably the most unexpected interaction we tested, and not one user discovered it.

Similarly, to continue reading once you hit the bottom of the screen might require any of 3 different gestures:

- Scrolling down within a text field, while staying within the same page
- For this gesture to work, you have to touch within the text field. However, text fields aren't demarcated on the screen, so you have to guess what text is scrollable.
- Swiping left (which can sometimes take you to the next article instead of showing more of the current article)

» This gesture doesn't work, however, if you happen to swipe within an area covered by an advertisement in The New York Times app

- Swiping up

iPad UIs suffer under a triple threat that causes significant user confusion:

- Low discoverability: The UI is mostly hidden within the etched-glass aesthetic without perceived affordances.
- Low memorability: Gestures are inherently ephemeral and difficult to learn when they're not employed consistently across apps; wider reliance on generic commands would help.
- Accidental activation: This occurs when users touch things by mistake or make a gesture that unexpectedly initiates a feature.

When you combine these three usability problems, the resulting user experience is frequently one of not knowing what happened or how to replicate a certain action to achieve the same result again. Worse yet, people don't know how to revert to the previous state because there's no consistent undo feature to provide an escape hatch like the Web's Back button.

Crushing Print Metaphor

Swiping for the next article is derived from a strong print metaphor in many content apps. In fact, this metaphor is so strong that you can't even tap a headline on the "cover" page to jump to the corresponding article. The iPad offers no homepages, even though users strongly desired homepage-like features in our testing. (They also often wanted search, which was typically not provided.)

In electronic media, the linear concept of "next article" makes little sense. People would rather choose for themselves where to go, selecting from a menu of related offerings.

A strategic issue for iPad user experience design is whether to emphasize user empowerment or author authority. Early designs err on the side of being too restrictive. Using the Web has given people an appreciation for freedom and control, and they're unlikely to happily revert to a linear experience.

Publishers hope that users will perceive content as more valuable if each publication is a stand-alone environment. Similarly, they hope for higher value-add if users spend more time with fewer publications rather than flit among a huge range of sites like they do on the Web.

Using the desktop Web, a user can easily visit 100 sites in a week, viewing only 1–3 pages on most of them. (For example, for one task in which B2B users visited 15 sites, they spent an average of 29 seconds per pageview.) Most sites are visited once-only, because users dredge them up in a search or stumble upon links from other sites or social media postings. Without real customer relationships, content sites have no value and 90% of the money created by users spending time online accrues to search engines.

The current design strategy of iPad apps definitely aims to create more immersive experiences, in the hope of inspiring deeper attachments to individual information sources. This cuts against the lesson of the Web, where diversity is strength and no site can hope to capture users'

> "Better to use consistent interaction techniques that empower users to focus on your content instead of wondering how to get it."

sole attention. Frequent user movements among websites has driven the imperative to conform with interface conventions and to create designs that people can use without any learning (or even much looking around). The iPad could be different if people end up getting just a few apps and sticking with them.

Card Sharks vs. Holy Scrollers

UI pioneer Jef Raskin once used the terms card sharks vs. holy scrollers to distinguish between two fundamentally different hypertext models:

- Cards have a fixed-size presentation canvas. You can position your information within this two-dimensional space to your heart's content (allowing for beautiful layouts), but you can't make it any bigger. Users have to jump to a new card to get more info than will fit on a single card. HyperCard was the most famous example of this model.
- Scrolls provide room for as much information as you want because the canvas can extend as far down as you please. Users have to jump less, but at the cost of less-fancy layout because the designer can't control what users are seeing at any given time.

The Web is firmly in holy-scroller camp, particularly these days: users scroll a fair amount and sometimes view information far down long pages. Even mobile-phone apps often rely on scrolling to present more than will fit on their tiny screens.

In contrast, card sharks dominate the early iPad designs. There's a bit of scrolling here and there, but most apps try to create a fixed layout for the pretty screen.

There's no real reason we can't have both design models: cards on the iPad and scrolls on the desktop (and phones somewhere in the middle). But it's also possible that we'll see more convergence and that the Web's interaction style will prove so powerful that users will demand it on the iPad as well.

Toward a Better iPad User Experience

Even our limited initial user studies provide directions for making iPad designs more usable:

- Add dimensionality and better define individual interactive areas to increase discoverability through perceived affordances of what users can do where.
- To achieve these interactive benefits, loosen up the etched-glass aesthetic. Going beyond the flat-land of iPad's first-generation apps might create slightly less attractive screens, but designers can retain most of the good looks by making the GUI cues more subtle than the heavy-handed visuals used in the Macintosh-to-Windows-7 progression of GUI styles.
- Abandon the hope of value-add through weirdness. Better to use consistent interaction techniques that empower users to focus on your content instead of wondering how to get it.
- Support standard navigation, including a Back feature, search, clickable headlines, and a homepage for most apps.

Although our full report offers additional detailed advice, we obviously haven't yet developed a full list of design guidelines.

One big question will remain unanswered for a year or so until we see how daily use of the iPad evolves: Will people use the iPad mainly for more immersive experiences than the desktop and mobile Webs? In other words, will people primarily settle on a few sources and dig into them intensively, rather than move rapidly between many sources and give each cursory attention?

Maybe people will begin to use the desktop Web for more goal-driven activities, such as researching new issues or performing directed tasks like shopping and managing their investments. And they might use the iPad for more leisurely activities, such as keeping up with the news (whether "real" news or social network updates) and consuming entertainment-oriented content. We don't know yet. The answer to this question will determine how far iPad UIs have to move from their current wacky style. ■

Jakob Nielsen, PhD, is principal of Nielsen Norman Group (www.nngroup.com), a user -research firm specializing in Web usability. He is the author or editor of 12 books, including the recent Eyetracking Web Usability (New Riders Press). Dr. Nielsen writes a bi-weekly newsletter, The Alertbox, with a quarter-million readers, at *www.useit.com*.

Reprinted with permission of the original author. First appeared in *http://www.useit.com/alertbox/ipad.html*.

On "Humans prefer cockiness to expertise"

*http://bit.ly/c4UR3b

From JONATHAN TANG (nostrademons)

Sounds crazy but I've used this in action. How do you think I got such high karma here? ;-)

Thing is - it works. Both online and in-person. I'd much rather be honest about how little I know (and often am when I'm working long-term with someone), but I've found it's a losing strategy in most situations. If you do know your stuff, you'll just get shouted down by idiots. Better to shout the idiots down first and then do the research to make sure you're not wrong. If you screw up everything, you'll probably get another chance simply by virtue of confidence (look at John Meriweather, who nearly brought down the global financial system three times and is still managing money), but if you appear timid and then screw up, people are all like "I knew he didn't really know what he was talking about..."

Answer to "What text editor do you use?"

From JOE COOPER (SwellJoe)

vim or emacs: pick one and get back to work. Editing text is a solved problem.

On "Online advertising is now dead"

*http://bit.ly/d9UK2e

From THOMAS PTACEK (tptacek)

The other day, Dave Winer broke his Cuisinart coffee machine and was, within 5 minutes, able to replace it on Amazon. Therefore, online advertising is now dead.

On "Fake Steve Jobs: Why I'm Switching to Android"

*http://bit.ly/aOA6qK

From ED WEISSMAN (edw519)

1980: CPM on your choice of hardware or lock into Apple hardware & software at a higher price.
1990: DOS on your choice of hardware or lock into Apple hardware & software at a higher price.
2000: Windows on your choice of hardware or lock into Apple hardware & software at a higher price.
2010: Android on your choice of hardware or lock into Apple hardware & software at a higher price.

Answer to "I'm Tired of Hacking. What Do I Do?"

From MAHMUD MOHAMED (mahmud)

I took a laptop and a digital camera with me and ended up hating them every step of the way. My first travels I did Africa and the middle-east, the second I did asia.

In countries where I have "based" myself, anything more than 4 weeks; the laptop has been a good useful distraction. When you're shocked by a local culture which you have to deal with for extended survival (anything more substantial than a western-style hotel and continental breakfast) you will end up missing speaking your familiar language, eating familiar foods, or just walking outside without a guide at hand (printed or in-flesh.) Also there is that strong sense of alienation when everybody around you is looking at you, even when you have been with them for weeks. In these times, firing up your slackware box and seeing what you used to work on in more homely times is a good psychological aid.

Cameras I didn't like. I hated being looked at and treated as a "foreigner", and I feel like I am doing the same when I point a lens at a "local" person, building or artifact. It felt like I was capturing their soul to take back home with me as a novelty. I have no photos of my travels, but I have friends. Hundreds of good friends from all walks of life; fishermen, priests, pimps, students, political activists, drug traffickers, aid workers, moms, bicycle repairmen, white-house staffers, journalists you name them.

Coming back was hard. I have lost 80lbs and came back with more street-sense than I could imagine. When I landed at Dulles Airport I had $60 to my name and I had the photo of a new girlfriend in my wallet. None of my family or friends had the time to give me a ride home, so I took the bus, for the first time in the U.S. Before then I have taken the bus a few times on nights-out when I knew I wouldn't be fit to drive. This time it was just what I was used to do. My instincts where different; I took a window seat in the way back that was close to an exit door. Something that you do when traveling in dangerous places (you don't sit in the front, or police and bandits will pull over the bus and shake you up for bribes; and you don't sit sandwitched between two locals, unable to escape.)

I also came back with 2pack a day cigarette habit. Hi alcohol tolerance. A very unprofessional appearance. An appetite for anything served to me on a plate. A habit of carrying a bag with basic survival necessities. Indifference to crashing anywhere. Hitching rides with total strangers. And finally, a weird ability to connect with people in the underworld.

My first few gigs have been freelancing gigs doing anything and everything. It took my girlfriend the last few months polishing up back to shape; I don't think I would have come back if it wasn't for her, actually. I have seen many long-time Western expats dying in local hospitals of controllable diseases; the ex-military Americans are most prone to this. Diabetes, high-blood pressure, liver problems; I have pitched in $5 donations to so many expats in hospitals I didn't want to be one of them.

On "Why Your Startup Shouldn't Copy 37signals or Fog Creek" *http://bit.ly/cfkZ4R

From MICHAEL F BOOTH (mechanical_fish)

This guy has gone to the zoo and interviewed all the animals. The tiger says that the secret to success is to live alone, be well disguised, have sharp claws and know how to stalk. The snail says that the secret is to live inside a solid shell, stay small, hide under dead trees and move slowly around at night. The parrot says that success lies in eating fruit, being alert, packing light, moving fast by air when necessary, and always sticking by your friends.

His conclusion: These animals are giving contradictory advice! And that's because they're all "outliers".

But both of these points are subtly misleading. Yes, the advice is contradictory, but that's only a problem if you imagine that the animal kingdom is like a giant arena in which all the world's animals battle for the Animal Best Practices championship [1], after which all the losing animals will go extinct and the entire world will adopt the winning ways of the One True Best Animal. But, in fact, there are a hell of a lot of different ways to be a successful animal, and they coexist nicely. Indeed, they form an ecosystem in which all animals require other, much different animals to exist.

And it's insane to regard the tiger and the parrot and the snail as "outliers". Sure, they're unique, just as snowflakes are unique. But, in fact, there are a lot of different kinds of cats and birds and mollusks, not just these three. Indeed, there are creatures that employ some cat strategies and some bird strategies (lions: be a sharp-eyed predator with claws, but live in communal packs). The only way to argue that tigers and parrots and snails are "outliers" is to ignore the existence of all the other creatures in the world, the ones that bridge the gaps in animal-design space and that ultimately relate every known animal to every other known animal.

So, yes, it's insane to try to follow all the advice on the Internet simultaneously. But that doesn't mean it's insane to listen to 37signals advice, or Godin's advice, or some other company's advice. You just have to figure out which part of the animal kingdom you're in, and seek out the best practices which apply to creatures like you. If you want to be a stalker, you could do worse than to ask the tiger for some advice.

Answer to "How do I become smarter?"

From MARK MAUNDER (mmaunder)

I have great news for you. The brain is extremely plastic. Read about neuroplasticity here: *http://en.wikipedia.org/wiki/Neuroplasticity*

Rest assured that your capacity to acquire new skills and knowledge is massive.

You don't just get smarter. You get smarter at something in particular. Playing chess, doing IQ tests, running the 100m dash, programming, social skills, public speaking, etc. So you need to pick a particular skill or set of skills or vocation and decide to get smarter at that.

There are some general rules for improving brain function though. Here are a few:

1. Read books. Reading trains your brain to concentrate for long periods of time without fatigue or distraction. There is a growing school of thought that the short bursts of reading and frequent distractions we experience online are harming our ability for deep contemplation, introspection and concentration. See Nicholas Carr, The Shallows. *http://n.pr/bnAfRV*

2. Try to get 10 hours of sleep a night. Sleep improves mental and athletic performance. *http://n.pr/9wQsXr*

3. Maintain your cardiovascular fitness. I highly recommend running. After years of cycling, swimming, hiking, etc I've found that running gives my brain function the biggest boost and provides me with sustained mental energy through the day. A good cardiovascular system supplies your brain with plenty of healthy oxygen rich blood. It's like putting racing fuel in your car.

4. Eat well. Cook your own food. Avoid processed or pre-prepared foods and non-organic foods (mainly due to the pesticides). Fish is awesome, but watch out for mercury.

5. Don't drink anything stronger than wine. Don't do drugs. (just like your mom told you)

6. Watch your weight. I find the biggest source of mental fatigue is when I've gained a few pounds.

Good luck, and congratulations on making the decision at a relatively young age to focus on your mental fitness.

On "If architects had to work like software developers"

* http://bit.ly/aA2FWB

From REGINALD BRAYTHWAYT (raganwald)

1. What makes you think Architects don't have to deal with fickle customers who have no concept of time, space, or budget?

2. Every project of any description needs a change control process. If yours consists of exchanging emails, it is going to go this way whether you're a web developer or a tailor.

3. The more expertise a customer thinks they have in the subject matter relative to you, the more comfortable they are micro-managing it. What have you done to educate the customer about how much expertise you bring to their project?

On Working Remotely

BY JEFF ATWOOD

When I first chose my own adventure, I didn't know what working remotely from home was going to be like. I had never done it before. As programmers go, I'm fairly social. Which still means I'm a borderline sociopath by normal standards. All the same, I was worried that I'd go stir-crazy with no division between my work life and my home life.

Well, I haven't gone stir-crazy yet. I think. But in building Stack Overflow, I have learned a few things about what it means to work remotely — at least when it comes to programming. Our current team encompasses 5 people, distributed all over the USA, along with the team in NYC.

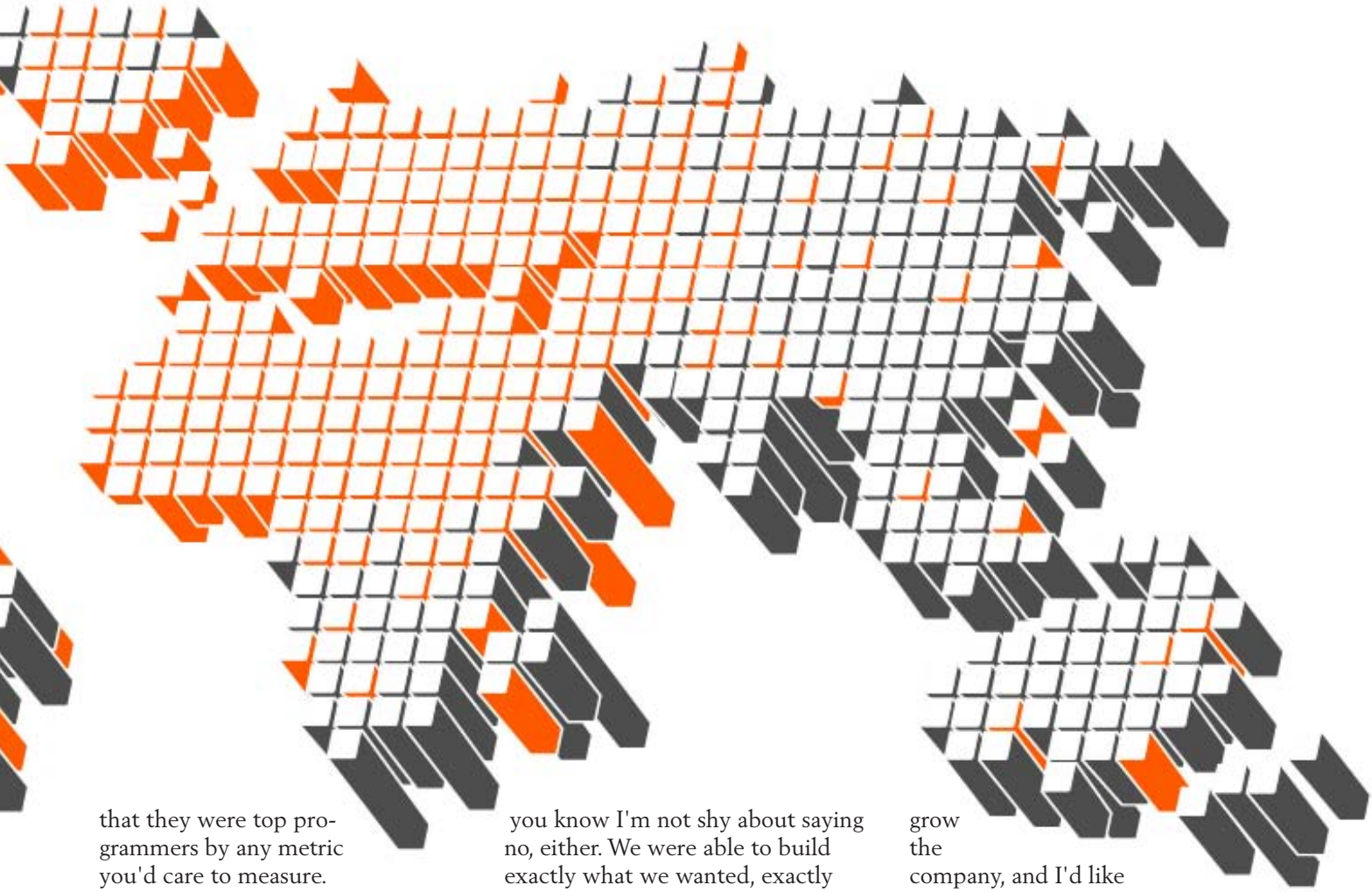
My first mistake was attempting to program alone. I had weekly calls with my business partner, Joel Spolsky, which were quite productive in terms of figuring out what it was we were trying to do together — but he wasn't writing code. I was coding alone. Really alone. One guy working all by yourself alone. This didn't work at all for me. I was unmoored, directionless, suffering from analysis paralysis, and barely able to get motivated enough to write even a few lines of code. I rapidly realized that I'd made a huge mistake in not having a coding buddy to work with.

That situation rectified itself soon enough, as I was fortunate enough to find one of my favorite old coding buddies was available. Even though Jarrod was in North Carolina and I was in California, the shared source code was the mutual glue that stuck us together, motivated us, and kept us moving forward. To be fair, we also had the considerable advantage of prior history, because we had worked together at a previous job.

> **"Always have a buddy, even if your buddy is on another continent halfway across the world."**

But the minimum bar to work remotely is to find someone who loves code as much as you do. It's enough. Anything else on top of that — old friendships, new friendships, a good working relationship — is icing that makes working together all the sweeter. I eventually expanded the team in the same way by adding another old coding buddy, Geoff, who lives in Oregon. And again by adding Kevin, who I didn't know, but had built amazing stuff for us without even being asked to, from Texas. And again by adding Robert, in Florida, who I also didn't know, but spent so much time on every single part of our sites that I felt he had been running alongside our team the whole way, there all along.

The reason remote development worked for us, in retrospect, wasn't just shared love of code. I picked developers who I knew — I had incontrovertible proof — were amazing programmers. I'm not saying they're perfect, far from it, merely



that they were top programmers by any metric you'd care to measure. That's why they were able to work remotely. Newbie programmers, or competent programmers who are phoning it in, are absolutely not going to have the moxie necessary to get things done remotely — at least, not without a pointy haired manager, or grumpy old team lead, breathing down their neck. Don't even think about working remotely with anyone who doesn't freakin' bleed ones and zeros, and has a proven track record of getting things done.

While Joel certainly had a lot of high level input into what Stack Overflow eventually became, I only talked to him once a week, at best (these calls were the genesis of our weekly podcast series). I had a strong, clear vision of what I wanted Stack Overflow to be, and how I wanted it to work. Whenever there was a question about functionality or implementation, my team was able to rally around me and collectively make decisions we liked, and that I personally felt were in tune with this vision. And if you know me at all, you know I'm not shy about saying no, either. We were able to build exactly what we wanted, exactly how we wanted.

Bottom line, we were on a mission from God. And we still are.

So, there are a few basic ground rules for remote development, at least as I've seen it work:

- The minimum remote team size is two. Always have a buddy, even if your buddy is on another continent halfway across the world.
- Only grizzled veterans who absolutely love to code need apply for remote development positions. Mentoring of newbies or casual programmers simply doesn't work at all remotely.
- To be effective, remote teams need full autonomy and a leader (PM, if you will) who has a strong vision and the power to fully execute on that vision.

This is all well and good when you have a remote team size of three, as we did for the bulk of Stack Overflow development. And all in the same country. Now we need to grow the company, and I'd like to grow it in distributed fashion, by hiring other amazing developers from around the world, many of whom I have met through Stack Overflow itself.

But how do you scale remote development? Joel had some deep seated concerns about this, so I tapped one of my heroes, Miguel de Icaza — who I'm proud to note is on our all-star board of advisors — and he was generous enough to give us some personal advice based on his experience running the Mono project, which has dozens of developers distributed all over the world.

At the risk of summarizing mercilessly (and perhaps too much), I'll boil down Miguel's advice the best I can. There are three tools you'll need in place if you plan to grow a large-ish and still functional remote team:

1 Real time chat

When your team member lives in Brazil, you can't exactly walk by his desk to ask him a quick question,

> "Chat is the most essential and omnipresent form of communication you have when working remotely."

or bug him about something in his recent checkin. Nope. You need a way to casually ping your fellow remote team members and get a response back quickly. This should be low friction and available to all remote developers at all times. IM, IRC, some web based tool, laser beams, smoke signals, carrier pigeon, two tin cans and a string: whatever. As long as everyone really uses it.

We're currently experimenting with Campfire, but whatever floats your boat and you can get your team to consistently use, will work. Chat is the most essential and omnipresent form of communication you have when working remotely, so you need to make absolutely sure it's functioning before going any further.

2 Persistent mailing list

Sure, your remote team may know the details of their project, but what about all the other work going on? How do they find out about that stuff or even know it exists in the first place? You need a virtual bulletin board: a place for announcements, weekly team reports, and meeting summaries. This is where a classic old-school mailing list comes in handy.

We're using Google Groups and although it's old school in spades, it works plenty well for this. You can get the emails as they arrive, or view the archived list via the web interface. One word of caution, however. Every time you see something arrive in your inbox from the mailing list you better believe, in your heart of hearts, that it contains useful information. The minute the mailing list becomes just another "whenever I have time to read that stuff", noise engine, or distraction from work … you've let someone cry wolf too much, and ruined it. So be very careful. Noisy, argumentative, or useless things posted to the mailing list should be punishable by death. Or noogies.

3 Voice and video chat

As much as I love ASCII, sometimes faceless ASCII characters just aren't enough to capture the full intentions and feelings of the human being behind them. When you find yourself sending kilobytes of ASCII back and forth, and still are unsatisfied that you're communicating, you should instill a reflexive habit of "going voice" on your team.

Never underestimate the power of actually talking to another human being. I know, I know, the whole reason we got into this programming thing was to avoid talking to other people, but bear with me here. You can't be face to face on a remote team without flying 6 plus hours, and who the heck has that kind of time? I've got work I need to get done! Well, the next best thing to hopping on a plane is to fire up Skype and have a little voice chat. Easy peasy. All that human nuance which is totally lost in faceless ASCII characters (yes, even with our old pal *<:-)) will come roaring back if you regularly schedule voice chats. I recommend at least once a week at an absolute minimum; they don't have to be long meetings, but it sure helps in understanding the human being behind all those awesome checkins.

Nobody hates meetings and process claptrap more than I do, but there is a certain amount of process you'll need to keep a bunch of loosely connected remote teams and developers in sync.

1 Monday team status reports

Every Monday, as in somebody's-got-a-case-of-the, each team should produce a brief, summarized rundown of:

- What we did last week
- What we're planning to do this week
- Anything that is blocking us or we are concerned about

This doesn't have to be (and in fact shouldn't be) a long report. The briefer the better, but do try to capture all the useful highlights. Mail this to the mailing list every Monday like clockwork. Now, how many "teams" you have is up to you; I don't think this needs to be done at the individual developer level, but you could.

2 Meeting minutes

Any time you conduct what you would consider to be a "meeting" with someone else, take minutes! That is, write down what happened in bullet point form, so those remote team members who couldn't be there can benefit from — or at least hear about — whatever happened.

Again, this doesn't have to be long, and if you find taking meeting minutes onerous then you're probably doing it wrong. A simple bulleted list of sentences should suffice. We don't

Reprinted with permission of the original author. First appeared in *http://www.codinghorror.com/blog/2010/05/on-working-remotely.html*.

need to know every little detail, just the big picture stuff: who was there? What topics were discussed? What decisions were made? What are the next steps?

Both of the above should, of course, be mailed out to the mailing list as they are completed so everyone can be notified. You do have a mailing list, right? Of course you do!

If this seems like a lot of jibba-jabba, well, that's because remote development is hard. It takes discipline to make it all work, certainly more discipline than piling a bunch of programmers into the same cubicle farm. But when you imagine what this kind of intellectual work — not just programming, but anything where you're working in mostly thought-stuff — will be like in ten, twenty, even thirty years … don't you think it will look a lot like what happens every day right now on Stack Overflow? That is, a programmer in Brazil helping a programmer in New Jersey solve a problem?

If I have learned anything from Stack Overflow it is that the world of programming is truly global. I am honored to meet these brilliant programmers from every corner of the world, even if only in a small way through a website. Nothing is more exciting for me than the prospect of adding international members to the Stack Overflow team. The development of Stack Overflow should be reflective of what Stack Overflow is: an international effort of like-minded — and dare I say totally awesome — programmers. I wish I could hire each and every one of you. OK, maybe I'm a little biased. But to me, that's how awesome the Stack Overflow community is.

I believe remote development represents the future of work. If we have to spend a little time figuring out how this stuff works, and maybe even make some mistakes along the way, it's worth it. As far as I'm concerned, the future is now. Why wait? ■

Jeff Atwood lives in Berkeley, CA with his wife, two cats, and a whole lot of computers. He is best known as the author of popular blog Coding Horror and the cofounder of Stack Overflow with Joel Spolsky.

Increase Conversion Rate by Making Your Site Ugly

By ZACK LINFORD

OVER THE YEARS many have contemplated the counter-intuitive ability of "ugly" sites to win huge market share – think eBay.com, Amazon.com, DrudgeReport.com, PlentyofFish.com, CraigsList.org, MySpace.com, or usability expert Jakob Nielsen's Useit.com.

In our adventures in website optimization we've developed our own grand unified theory of why ugly web design works:

> **"We trust things more when they look like they were done for the love of it rather than the sheer commercial value of it."**
>
> - Robert Scoble

1 Value – Your visitors want a deal. Never, never, never forget that.

We're a nation of Walmart shopping, McDonald's value meal eating, 2-Buck Chuck drinking coupon-clippers.

If your website looks BMW-fancy your visitor is going to assume BMW-pricing.

Make your visitors think that they've found the last great deal – look a little pathetic and rough around the edges and your visitor is going to assume that they're not going to be taken advantage of.

2 Trust – Nobody likes advertising, or advertisers (except their wives).

Advertising ranks amongst the LEAST respected professions and most people strongly dislike being advertised to because they feel manipulated.

Eliminating stock-photos, fancy graphics, and high-brow design elements can help your cause and make you feel more ma & pa trustworthy than a corporate-titan in training.

3 Accessibility – Build for technology two cycles back.

HTML5, the latest CSS tricks, and your kickass integrated flash design have NO PLACE in a website designed to sell when older technologies can do a comparable job.

One of our clients receives in excess of 15,000 visitors a day to their website – about 70% of that is coming from various versions of Internet Explorer.

	Browser Version (None)	Visits ↓	Visits
1.	8.0	11,719	74.14%
2.	7.0	3,022	19.12%
3.	6.0	1,056	6.68%
4.	5.5	10	0.06%

Yet nearly 27% are using outdated versions despite wide availability.

So unless you enjoy building 10 versions of your site stick with simple and build for compatibility with browsers, OS, screen resolutions, color palettes, etc.

4 Flexibility – Don't paint yourself into a corner.

What do PlentyofFish, CraigsList, and DrudgeReport have in common?

They scaled to huge numbers of visitors with tiny staffs – keeping your site flexible enough so the CEO can change the homepage content may not be aesthetically appealing, but it sure

Reprinted with permission of the original author. First appeared in *http://www.conversionvoodoo.com/blog/2010/04/increase-your-conversion-rate-by-making-your-site-uglier/*.

does beat a static beautiful website.

A website that's easy to change, update, and experiment on is better than one that relies heavily on advanced CSS, Flash, images etc that you can't change quickly.

5 Function – Get your users where they want to be as your priority.

When you're running a commercial website just by virtue of having arrived, a user is a qualified visitor ready for you to close.

So get the !@%$!@% out of their way and let them transact!

Keep it simple

- Make sure your homepage is crystal clear to let a user determine if your website will fulfill their need.
- Let users get where they need to go in as few clicks as possible.

Any design element that detracts from your focus – will lose the user – one of my favorite examples of this is from a Marketing Experiments study on email:

MarketingExperiments Email Designs			
Email Designs	CR	Rel. Diff	Revenue
Version A	17.60%	---	---
Version B	**24.38%**	**38.49%**	**61.42%**
Version C	19.85%	12.73%	33.52%

Of the three emails above B outperforms the other two design-element laden tests by 62%!

It's no surprise that the winning test lacks over-blown design elements & complexity, keeping it simple collects the sale.

We've battled designers and CMO's day in and day out for nearly a decade but overwhelmingly following the 5-rules laid out above drive results that simply win. ■

Zack Linford is the co-founder of ConversionVoodoo.com – a company dedicated to increasing website conversion rates.

Zero Zero

By RAFAEL CORRALES

I'VE BEEN THINKING about something that we always did junior year when I was on my high school soccer team.

When we'd score a goal, we realized that it's when a team is at its most vulnerable. I saw it first hand when many earlier teams I had been on would get scored on right after our goal. It negates the whole point of working so hard for that score.

So that year, after a goal, we would pause and celebrate for just a few seconds. And as we ran back to our side of the field, we always had one guy stop and yell at the top of his lungs, "WHAT'S THE SCORE?" and we'd yell back "ZERO-ZERO!"

That scared our competitors, but more importantly it got us results. That year we outscored the teams we played something like 48 goals for and 6 against. We beat some of the best teams in the southeast and some really big schools.

My varsity team was barely 15 guys from a 200 person school. We had a high concentration of really talented people, but the big part of our success wasn't our talent. Our success was actually the result of our mentality. And that's the broader point: don't let your success turn into complacency. Because right after a small success is when you are the most vulnerable to complacency and bad results. ■

Rafael Corrales is co-founder and CEO of LearnBoost, a VC and angel backed education startup offering free gradebook software. He graduated from Georgia Tech and holds an MBA from Harvard Business School.

Reprinted with permission of the original author. First appeared in *http://blog.rafaelcorrales.com/2010/05/zero-zero.html.*

Mistakes I've Made & What

By JAQUES MATTHEIJ

I've been running my own companies since 1986. That's 24 years now, with some brief stints of employment if a contract was so time consuming that the dutch regulators took it as being equivalent to employment (they do that here to stop employers that try to avoid paying in to social security by hiring all their employees as free-lancers). At the high point of running 'TrueTech' we had about 20 full timers and partners, and a bunch of free-lancers.

It's been a long, very interesting and at times very stressful ride so far, and I wished I could say I never made any mistakes.

But I have. Plenty of them, and most of them seem to be related to personality traits, I've tried to outline those below.

Some of the mistakes were almost without consequences, some of them with grave consequences. Here are the 'highlights', hopefully they'll save some of the readers of this from repeating them.

I'm fairly gullible and I tend to believe that what people tell me is true

I don't usually follow up to verify that what I'm told is true, I was raised in an environment where almost everybody simply spoke the truth.

Automatically I assumed (and it seems to be a pretty strongly ingrained thing, I still have this today) that everybody is always truthful.

That's a serious weak point, and it has cost me dearly on a few occasions. Over time I've become more wary, especially the last 15 years have shown me a few very nasty instances of how cunning and calculating people can be when they deceive those around them for profit. I've gotten a lot better at spotting inconsistencies in peoples' stories and this has helped to mitigate the gullibility factor to some extent, but if someone comes to me with a sob story I'm more likely than not overwhelmed by the emotion and willing to help even when I really should be more cautious. And every now and then a sob story is real, even when it sounds highly unlikely.

This particular mistake has cost me dearly over the years and has changed my personality to someone that is much more cautious than he would like to be.

When evaluating people I always see the potential, but hardly ever the reality

Most people achieve only a fraction of what they could do theoretically.

My problem is that when presented with a potential employee or partner that I tend to see what they *could* do, but not what they actually realistically speaking will be able to do.

It's like looking at a sports car, you know it can do 150 miles per hour, but in real life circumstances it will hardly ever do that, more likely it will just have to obey the usual traffic rules and will periodically need refueling and so on. So you have to adjust your expectations based on real world conditions, and I'm very bad at that.

If it hadn't been for that I could have predicted the burning out of some people in my surroundings with greater accuracy and possibly I could have prevented it from happening, and I would have been better able to estimate how much work I could expect to get out of a given configuration of people working on projects.

You Might Learn From Them

I either delegate too much or too little

This is probably one of my biggest shortcomings, when delegating stuff I either hand it off and don't look back until I'm presented with some kind of disaster, or I'm so on top of it that whoever is doing the job feels like the dragon is breathing down their neck all the time. The sweet spot is somewhere in the middle, but I haven't found it yet.

Over the years this has made life harder for employees, partners and customers, I could have done a *much* better job here. Trust but verify is something that I heard about way too late, it also applies to some extent to '1' above. The mistakes I made because of this are along the line of letting people run a subsidiary company for over 3 months without checking the books (and finding out much too late that they'd gone off and spent 3 months worth of turn over in the local casino!), or riding shotgun on a new developer and disagreeing with just about every thing he did only to find out many years later that there are multiple equally valid solutions to a problem. This is probably one of the hardest things for me to do, to 'let go' and to accept that someone else will do something different from the way I would do it, but will still do a good enough job of it.

I'm a loner

When it comes to doing things, I can do way too much. Electronics, basic engineering, software, metalworking, woodworking and so on. If there is a technical skill I've probably tried my hand at it, and can do a reasonably job of it. Not perfect, but good enough for government work. That means that I'm pretty self sufficient and there are only a few fields where I know that I absolutely suck. On top of that I'm a voracious reader with an extremely wide interest, I remember most of what I've read.

Higher mathematics and design would be two of the fields that I really suck at, as well as managing people. The result of that is that I was pretty happy running my one man company and *completely* not prepared to deal with the reality of growing it, more people.

I wished that the 'school of life' up to that point had forced me more often to work together with people in a real team setting, first as a team member, and then as a team leader, so that I would have been better prepared to deal with that. It definitely didn't help in my relations with the employees of the company when it grew. I was just a 'techie', never planning to be in charge of a company that size and I grew in to the job very reluctantly.

Now I'm back to 'square 1', alone (or, more precisely with one business partner) and much more happy because of that, still not sure if I've learned these lessons well enough to be able to grow again. Maybe.

I have a lot of energy, but not everybody is like that

Another one of those 'expectation' issues, I can work on stuff with tremendous energy, but that's a rarity, and most people go about life at a more relaxed pace. I'm *always* doing something, I really can't sit still for more than 3 minutes without having to get up and doing something (unless I'm watching a movie or reading a book, but that's still doing something). Subconsciously I expect other people to be like that too, and I'm often quite surprised when they're tired or zoned out in front of the TV or simply doing nothing.

So I tend to burn people out, they try to keep up and give up after a while. I should try to slow down a bit to a more moderate pace and keep the 'energy bursts' to myself.

> **When I see stuff I do not agree with I am very outspoken, diplomacy is definitely not my strong suit.**

I have a short attention span

It is difficult for me to stay focused on the same thing for a long time. This started when I was a kid, if I got some new toy I would play with it for its intended purpose for about 10 minutes, then rip it apart to see how it worked. It took a long time before I had skills enough to put stuff back together again.

I still have this, I learn pretty quickly, but once I understand how something works the mystery has gone out of it and I am likely to move on. But give me a puzzle that is 'unsolvable' and I'll probably spend a lifetime on it.

The only exceptions here were Lego (I played with it over and over again), Electronics (taking stuff apart was both a source of parts and a way to learn) and programming.

I have to work really very hard to overcome this tendency and I'm pretty sure that it has cost me over the years to find little or no interest in doing the 'grunt' work of running a business.

I'm pretty harsh

When I see stuff I do not agree with I am very outspoken, diplomacy is definitely not my strong suit. Not everybody can deal with this and even though I try very hard to moderate the force I find it very difficult, especially when I think people are not nice to other people. That can bring out a force 7 gale in no time at all. Even though the emotion driving that is pure I could do a lot better by tempering my feelings and coming up with constructive criticism instead of full blown confrontation. This has soured my relationships with people on more than one occasion, and some of those people were important players in or around my business.

I take full responsibility for each and every mistake I've ever made, no matter whether or not other people were involved, if there was something that I could have done better then I regret not having done that. In the long term though, I hope I can improve these aspects and that by learning from my past mistakes which taught me about these traits, and I hope that I can avoid future repetitions.

I also hope that by reading about this you may be able to avoid some of my past mistakes. ■

Jacques Mattheij is the inventor of the live streaming webcam, founder of camarades.com / ww.com and a small time investor. He also collects insightful comments from Hacker News.

Reprinted with permission of the original author. First appeared in *http://jacquesmattheij.com/Mistakes+I've+made,+and+what+you+might+be+able+to+learn+from+them.*

Tell us what you think

Let us know what you liked, and what we need to work on.
Please share your thoughts so we can improve the coming issues.

hackermonthly.com/feedback/