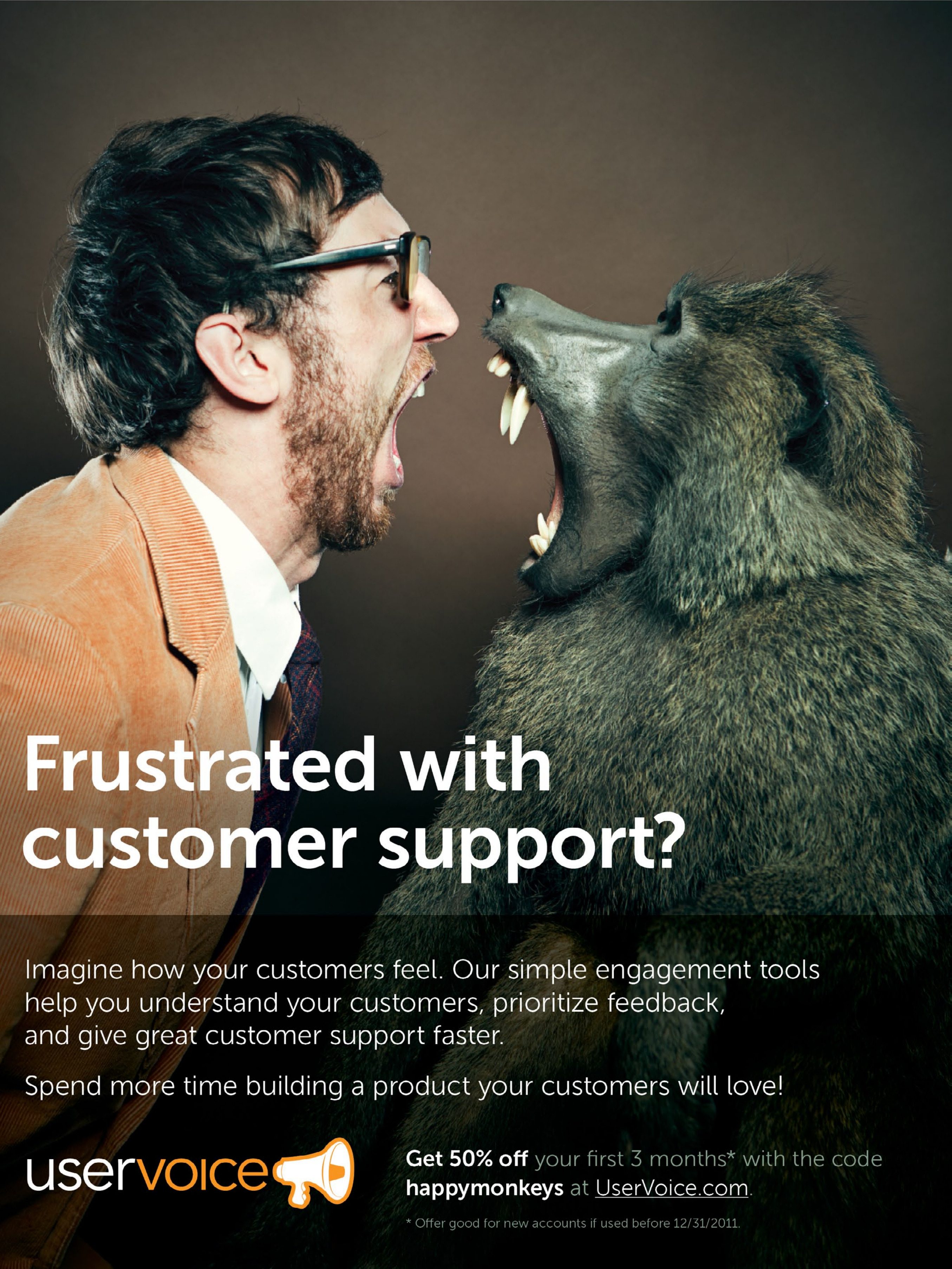
By Steve Yegge

What Would You Do With Your Own Google?

HACKERMONTHLY

Issue 17
October 2011



Frustrated with customer support?

Imagine how your customers feel. Our simple engagement tools help you understand your customers, prioritize feedback, and give great customer support faster.

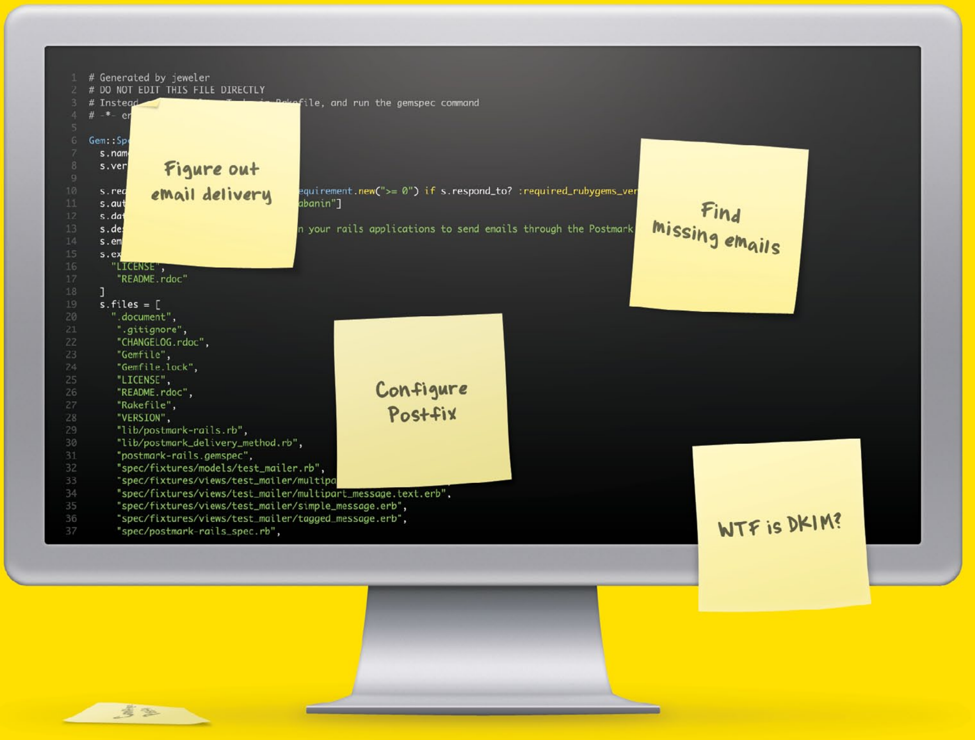
Spend more time building a product your customers will love!

uservoice

Get 50% off your first 3 months* with the code **happymonkeys** at UserVoice.com.

* Offer good for new accounts if used before 12/31/2011.

If you enjoy configuring Postfix and Sendmail, ignore this.



```
# Generated by jeweler
# DO NOT EDIT THIS FILE DIRECTLY
# Instead                        efile, and run the gemspec command
# -*- en

Gem::Spe
  s.nam
  s.ver

  s.req                          equirement.new(">= 0") if s.respond_to? :required_rubygems_ver
  s.aut                          abanin"]
  s.dat
  s.des                          n your rails applications to send emails through the Postmark
  s.em
  s.ex
    "LICENSE",
     "README.rdoc"
  ]
  s.files = [
    ".document",
     ".gitignore",
     "CHANGELOG.rdoc",
     "Gemfile",
     "Gemfile.lock",
     "LICENSE",
     "README.rdoc",
     "Rakefile",
     "VERSION",
     "lib/postmark-rails.rb",
     "lib/postmark_delivery_method.rb",
     "postmark-rails.gemspec",
     "spec/fixtures/models/test_mailer.rb",
     "spec/fixtures/views/test_mailer/multipa
     "spec/fixtures/views/test_mailer/multipart_message.text.erb",
     "spec/fixtures/views/test_mailer/simple_message.erb",
     "spec/fixtures/views/test_mailer/tagged_message.erb",
     "spec/postmark-rails_spec.rb",
```

We'll deal with that so you can get back to coding.

Postmark

hacker.postmarkapp.com

Transactional email delivery for web apps.

Curator
Lim Cheng Soon

Contributors
Rahul Bijlani
Steve Yegge
Kapil Kale
Purin Phanichphant
Yann Esposito
James O'Beirne
Sean Plott

Proofreader
Emily Griffin

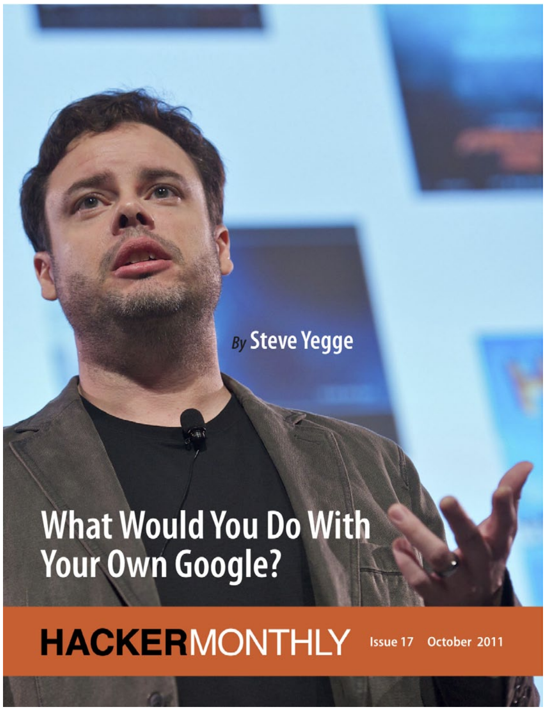
Printer
MagCloud

Advertising
ads@hackermonthly.com

Contact
contact@hackermonthly.com

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

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



Cover Photograph: James Duncan Davidson

Contents

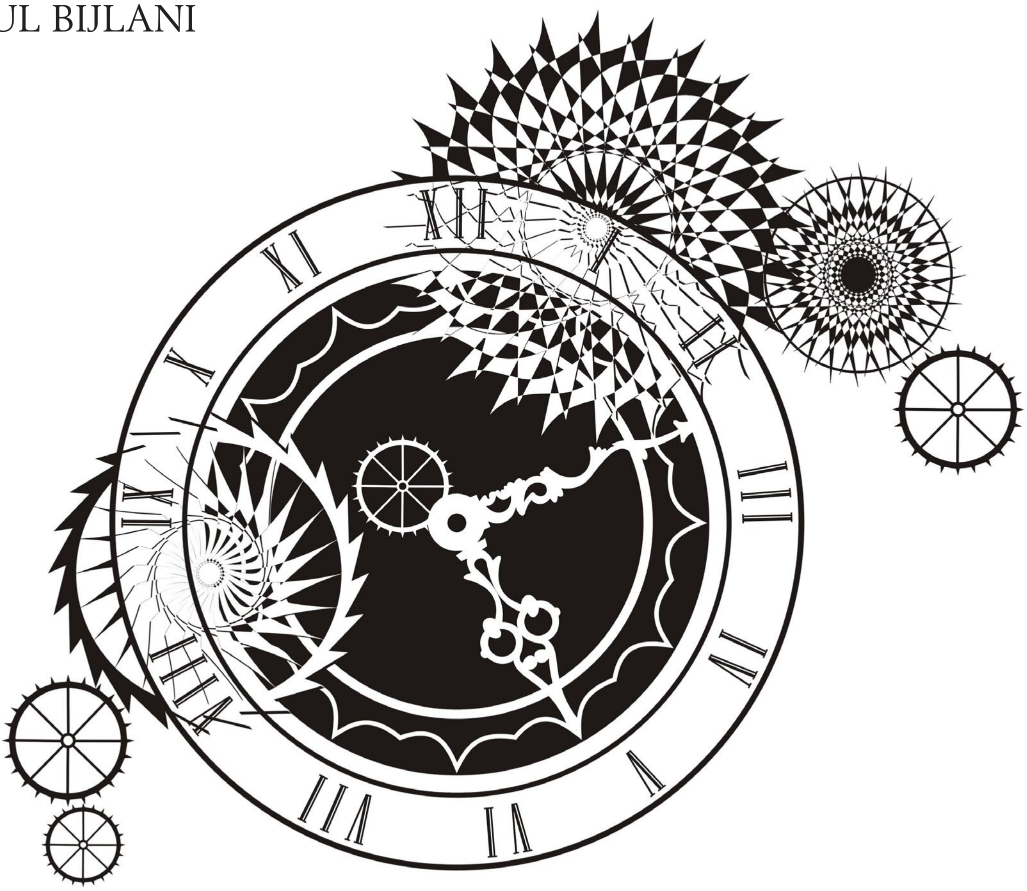
Photo courtesy of O'Reilly Media and James Duncan Davidson.

For links to the posts on Hacker News, visit *hackermonthly.com/issue-17*
All articles are reprinted with permission of their original author.

You Are Not Running Out of Time

How I Learned to Stop Worrying and Began Enjoying Infinity

By RAHUL BIJLANI



EARLY IN HIS political career, Julius Caesar is said to have wept upon reading a biography of Alexander the Great. When asked why, he apparently said, "Do you think I have not just cause to weep, when I consider that Alexander at my age had conquered so many nations, and I have all this time done nothing that is memorable!"

This story was seared in my memory when I read it in high school, because it spoke to my own search for achievement: I had read that at seventeen, Bill Gates had already created his first successful business venture. At the same age, I hadn't even figured out where to start. It didn't make me weep, but it did make me worry.

And so, incredibly, at seventeen, I genuinely wondered:

Was I running out of time?

It seems amusing now — but back then I was deadly serious.

The Game

You know the feeling — the feeling of being left behind in the race for achievement. Of falling back in "the game." For some people, the game is keeping up with the Joneses: marrying a good catch, living in a nice house, driving the right car, having a good job, kids that do well at school. For others, it is enjoying life's pleasures: the best vacations, the most enjoyable parties, with the most exciting partiers. Then there are people who are forever pursuing harmony and peace in their lives, resolving the discordant threads one by one. And for some the game is living up to their objective definition of personal development.

For most, it is a combination with a common thread: am I moving up in the world at an acceptable pace, or am I running out of time? Am I maximizing my potential?

What that quickly meant to me was that wasting time and opportunities were criminal, with my own potential achievements as victims that needed to be rescued from the assault of lost hours and non-productivity. It meant becoming a workaholic. Bill Gates probably felt that way once. Looking back at his teenage years and his own obsessive time spent with computers, he said,

"It was hard to tear myself away from a machine at which I could so unambiguously demonstrate success."

I thought I was on the right track.

“How did Bill Gates know he wasn’t running out of time to achieve his true potential when he made his first billion?”

A Moving Target

Ironically, when I started to cross some of my own personal benchmarks, I discovered that something was very wrong — I kept moving the goalposts.

One counter-intuitive handicap of playing the game is that with every step you move forward, two things happen:

1. You discover that it’s possible to go further than you previously knew.
2. The people you are left playing with are better at the game than people left behind. In other words, distinguishing yourself from your peers gets tougher as your definition of your peer group gets upgraded. It must have been easy for Bill Gates to stand out at Harvard, not so much in Silicon Valley, where he has constantly competed with Steve Jobs, Larry Ellison and others master games-men.

That’s why the “acceptable pace” aspect of moving up in the world keeps evolving as you discover greater and greater opportunities. When Bill Gates made his first million, it probably felt extraordinary to him, a landmark achievement. How about his second? His 20th? His 100th? How did he know he wasn’t running out of time to achieve his true potential when he made his first billion? If he was measuring himself on market domination, where would he go after 95% market share was secured?

The questions I had got crazier, but they seemed logical progressions of understanding the game. For example, geneticists say that one in twelve Asian men is descended from Genghis Khan. How did Julius Caesar feel about not leaving behind his empire to his progeny? Or Alexander for not having any children at all? Does that mean Genghis Khan played the game better? How does that make Bill Gates feel about marriage and kids? Does it make sense for him to have a harem, for example? Would it make sense for me to have one? And one child showered with attention, or the risk spread over a couple hundred?

“Playing the game the right way isn’t good enough. It needs to be played for the right reason.”

If you keep asking these questions, how can you not keep moving the goalposts? How can you not get exhausted, overwhelmed, or anxious?

The Journey

Eventually, I came across a thought from “Zen and the Art of Motorcycle Maintenance.” In the story, Pirsig, a young man, goes mountain-climbing with some elderly monks. He struggles throughout, and eventually gives up, while the monks easily continue to the peak. What is apparent is that Pirsig, focused as he is on the peak, is overwhelmed by the climb, and continues to lose his desire and strength with every step. The monks, on the other hand, used the peak only as a guide to mark the direction of their climb; they were more focused on the journey and its enjoyment, and made it to the top with ease.

This offered a valuable insight. Maybe Bill Gates doesn’t sit and ponder these definitions of success. Maybe he keeps it simple — to maximize his fortune and have a small, loving family — and simply enjoys programming. Maybe Alexander simply enjoyed battles, and Stephen Hawking loves physics. It would appear that they would still be active in those pursuits regardless of the relation of their endeavors to material success.

This would also suggest that the game, i.e. maximizing your potential, and what you can achieve with your time and resources, is best played if you enjoy the pursuit of your goals. In other words, if you are journey-based, rather than destination-driven. Pirsig’s monks probably just liked walking in the mountains, maybe they were not as wedded to the idea of standing on a peak as they were to enjoying nature.

“If you know what you want to build and play the game to enjoy the journey, you are probably on your way to the good life.”

Earlier, to me the game meant maximizing your time and potential to get somewhere. Now it means maximizing those things to enjoy the trip. That would mean that Bill Gates’ measure of success is how much he enjoyed his day, not how much code he wrote, or how much his businesses expanded.

A revolutionary thought! The point of my life was to enjoy it to its potential, with goals to set the direction in which I was headed.

This was my new definition of the game.

And it meant it was impossible to run out of time, because every day was a brand new opportunity to play and win.

But that still begged the question: how do you pick your destination? Doesn’t it keep moving, every time you re-evaluate the meaning of success? The monks had a fixed peak in the mountains they were climbing, most of us don’t have the luxury.

The Right Question

The answer to these questions occurred to me somewhat unexpectedly, through the best line in an otherwise unremarkable movie.

In Wall Street 2, right after he has cheated his own daughter out of her trust fund, Gordon Gekko, Hollywood’s favorite bad guy, is confronted by his future son-in-law, who chastises him for his seemingly slavish devotion to money. Gordon hears him out, and responds,

> *“You never did get it, did you? It’s never been about the money — it’s about the game!”*

While the audience shook its head in disapproval, a lifetime’s worth of questions were answered for me in a flash, and I wanted to jump up and cheer. I had the answer: Gordon was playing the game exactly right, and that’s why he was exactly wrong!

A wise man once said, “happiness is the ultimate currency.”

He wasn’t running out of time, and he genuinely enjoyed every day of playing the game. He didn’t even care about the money, which he made and lost and made back. And yet, he was unhappy and it was clear that something was very, very wrong.

What I realized was that playing the game the right way isn’t good enough. It needs to be played for the right reason: to build something, to see something grow. Gordon wasn’t building anything at all, not even a family, and his emptiness showed dramatically.

The Destination

The answer to how you pick the destination: by asking yourself, what do I want to see grow? What do I want to build?

Even Bill Gates seems to have an opinion on this. “I’m a great believer that any tool that enhances communication has profound effects in terms of how people can learn from each other, and how they can achieve the kind of freedoms that they’re interested in.” And sure enough, he’s been building these tools all his life. All the money he made doing it? He’s giving it away. And he’s enjoying that process, too!

Einstein wanted to build a theory that unified the physics of very large objects, like planets and the physics of very small objects, like atoms. Did he complete his project, before he died? No, but he left a legacy and a foundation for generations of future scientists to keep building on. I doubt he felt like he had run out of time.

A couple years ago, Steve Jobs built a phone that he wanted to see exist, and changed the world forever. Did he really need the money? Or the influence? Or the acclaim? Or was he simply trying to create something, and enjoying the process of seeing his vision come to life?

All of these examples suffered numerous setbacks as well as many opportunities to retire early in life, but they chose to keep moving because of what they wanted to build. If you know what you want to build and play the game to enjoy the journey, you are probably on your way to the good life. All of a sudden, the "am I running out of time?" question becomes meaningless.

Imagine building a house; would you really want to rush it? Imagine you faced an interruption, perhaps a snowstorm halted construction for a week. Would it make sense, or even be safe or wise to continue at the same pace during the storm? You wouldn't feel bad about the delay, you'd just wait till you could resume. Or imagine you ran out of funds. Would you abandon the project because it was running behind, or would you find a way to continue in the future? If the foundations were poured and then you were diverted for a year, would you consider the construction to have moved backwards, or merely paused?

Now imagine building a family, or a skill set, or any object or business. Is it more important to do it rapidly and compare it to others, or to build something that will last and bring your vision to life?

A Recipe For Life

These questions are also why comparisons don't really make any sense. Julius Caesar was weeping for all the wrong reasons. Alexander and he had different visions; they were looking to build different things in different times. Similarly, it was meaningless for my seventeen-year-old self to measure myself against a very different person's desires at a completely different time and place. In doing so, I was denying my own dreams, and trying to live someone else's. I was also assuming I knew what their dreams were in the first place. Maybe all Bill Gates was trying to do at seventeen was impress his high school crush. Maybe Alexander was trying to live up to the dreams of his father. The reality is that nobody will ever know!

Work, spouse, kids, and family are not items to be checked off a list. They are directly based on the vision of the life you are trying to build, and settling based on a clock is merely a guarantee that the vision is being compromised. On the other hand, realizing what you want to build, as opposed to solely playing the game, may dramatically impact the choices you make.

In fact, answering the "what do I want to see grow" question impacts all decisions, from what to do on a Saturday afternoon, to whether you should move to a different city for your work. It makes short-term and long-term destinations clear, and then all that is left is to play the game, or maximize your potential, to enjoy the journey of getting there. It also explains why the Gordon Gekkos and Julius Caesars of the world, who play the game just for its own sake, are generally unhappy and unsuccessful in their own eyes, even though they appear to be doing everything right.

A wise man once said, "happiness is the ultimate currency." The phrase resonated with me, but "the game" didn't help me maximize the currency that mattered most. Now however, at thirty, I think I have the ultimate business plan, and nobody is running out of time any time soon. ■

Rahul Bijlani brokers, owns and operates hotels. He's also part of a few technology startups, including Spinofix and YB Intel. He still does a little coding in his spare time.

What Would You Do With Your Own Google

By STEVE YEGGE

I'm not really a data guy. For the last four years or so, I've been working on stuff that I love. I want you to remember this, though — this is really important: you don't want to put out mediocre work, and you don't want to feel mediocre about what you're doing. Surprisingly, we don't always do this, though.

Right now, I work at Google, and for the last four years at Google (I've been there for six and a half total) I've been working on compiler technology, because I had this passion around developer tools and I've been working on static program analysis. So, when you take Google's source code and you treat it as data, you take all of the world's open source code and you treat it as data, it's actually not a very big data set. It's a couple of dozen terabytes, couple of hundred terabytes, but it's pretty tiny compared to the big data sets at Google.

I've had scaling problems, but I'm not really a data guy, per se. But, gosh, I work with people who do have data problems. I have to say, just in passing, Google is an awesome place to work. I went to work for Google actually, specifically, because I think that Google is trying to change the world, and they're serious about it. I kind of thought this when I went there six and a half years ago. And after six and a half years, Google is the only one trying to defend net neutrality, trying to open up China, when nobody else is — I mean that is pretty audacious.

They haven't always been successful in their goals. They're really trying to change the world, but they have big scaling problems. Sometimes it feels like it's their biggest problem. But it's funny, because scaling is sort of a periodic problem. You need to scale, because you've got a problem to solve, and then it's good enough for a while, and you can focus on problem domains. Then it becomes a problem again as you grow, just like software engineering scaling has been a problem kind of on and off. We're actually pretty good at it right now, whereas ten or fifteen years ago, we were building software systems that we didn't understand. So, we knew we needed to step back and get some new abstractions.

Anyway, before I was at Google, I was at Amazon.com, another company that's trying to change the world. Jeff Bezos has "grand visions" — it's another company with huge scaling problems. Now they are also awesome. It's kind of a different flavor of awesome than Google. They're pretty open about their work environment being very frugal, because their vision is to pass savings on to customers, but I wasn't there for the work environment. I wouldn't have stayed there six and a half years. They have great people there, and they have tremendous scaling problems.

“Hollywood calls them auteurs, but what they really are is people with principles who are making money.”

Scaling really is too often the biggest problem. Why? Because these companies are fundamentally relational and transactional. I know what you’re thinking. Nobody likes relational anymore, but they have to make it scale. And they succeed at it. Google also has some transactional data. Google uses MySQL for their ads system, and it’s really, really hard to scale, but you can make it happen.

Again, Amazon is a company that’s not mercenary. They’re trying to make money. And that’s good, because money is the fuel for innovation. But they’re doing it with principle. The retail thing that Amazon did was sort of like a means to an end, just like scaling is a means to an end. Now they’re doing cloud computing. Bezos has got big plans just like Larry and Sergey. So, what about the rest of us? What are we doing right now? What are we focused on?

Well, I want to look at one of our sister industries: Hollywood. It’s not exactly been a banner year for them, has it? You might have noticed all the movies suck this year — 28 sequels and comic book movies. Why is this? Well, we know the reasons. It’s human nature. It’s corporate greed, companies being mercenary, taking advantage of consumer apathy. If we keep paying to go see movies like “Scream 4” and “Hangover 2” and “Transformer 17” and whatever, then they’ll keep making them.

And the same thing’s going on in the game industry. Game industry has hundreds and hundreds of titles coming out. Maybe you like video games, or you have kids who like video games. You have friends who like video games and they’re always complaining that there aren’t any good games — but there are so many good games! There are talented people working on them, great animators and great cinematographers and sound people. Yet most of the games are pretty bad. Again, it’s because people keep paying for them, so people keep making them.

“Here’s an interesting problem that we could be working on, if we were literate: the Human Genome Project.”

Yeah, except for these guys. Hollywood calls them auteurs, but what they really are is people with principles who are making money. Either the principles are stretching you a little bit and making you think about philosophy and the nature of consciousness while you’re getting entertained, or they’re bringing up social issues like trashing the world in Wall-E, or they’re putting some serious history into their game experience like Rock Star. Whatever it is that they’re passionate about, it really bleeds through. It shows through in their final product, and they’re making everyone else look bad. They’re also showing that you can still make a ton of money with principle, and Jeff Bezos and Steve Jobs fall into this category. Anybody can have principle.

Bill Gates does today. He’s sort of atoning for his past sins. Did he have principle when he was at the helm at Microsoft? No, obviously, and now he knows well, he probably didn’t need to be that way.

Is social networking principled? What developer or entrepreneur is not working on something that’s kind of related to a social network? Seriously. Well, it’s fun and also obviously makes money. You can make a buck with cat pictures.

Social networks they have a purpose, and Facebook’s purpose is whatever they make easiest, because that’s what people are going to do with it. Facebook and Facebook clones, if anybody’s making any of those, are really good at sharing cat pictures. Now, they can have secondary benefits to humanity, like for example, helping out with the information flow into and out of the Middle East. But that’s not what they were designed for, so that’s not what people are using it for, and it winds up being crap just like Hollywood’s movies.

So, here’s what happens over time. As you get older, you start getting interested in issues broader than cat pictures. I’m not saying that you’re not interested in socializing anymore.

“If we just focus on scaling, we’re going to scale up FarmVille. It’s going to be Farm Planet.”

Obviously, you want to hang out with old people. But you also get interested in why they haven’t cured heart disease yet. As you gain wealth and if you’re not a total mercenary, then you start getting interested in charity and helping people who are needy. You start getting interested in politics and all these other kind of hard problems, and a lot of them interestingly are data mining problems.

Well, the problem is, by the time you’re old, it’s too late. What you want is a time machine, so you can go back and tell yourself, “Hey, man, if you had studied some math, then you could probably work on some of the hard problems like signal processing.” Voice recognition is an important problem, because it’s an accessibility thing. It’s putting blind people and deaf people in the same position as us. Natural language processing: same thing. Genes, viruses, there is a bunch of problems out there that have the common characteristic that they’re not just computing problems. So, they’re a little harder, because you’ve got to know a little bit of math, a little bit of statistics. You don’t have to be Hal Varian. You just have to be fluent. You have to be literate.

Well, here’s an interesting problem that we could be working on, if we were literate. Have you thought about this project at all? The Human Genome Project, what is it? The genome sequence that they’ve got now is the compiled binary to the source code for the human body, for life, the mechanics of life. If we had the source code in our hands, that would be the cure for cancer right there. That’d be the cure for all the viruses — all of them. That would keep antibiotics ahead of bacteria. Innumerable benefits. We know this. You could also hack things, or grow your own tattoo! It will be an inflection point in human history — and it’s a data mining problem.

“You’re only one step away from solving hard problems, important problems, world-changing problems.”

Now, you may already be good at data mining. There was a dude sitting next to me at dinner last night named Jay who was a freaking superhero. He is a thousand times smarter than I am. I brought up this project, and he says, “I’m not really a bio-informatics guy, but you know…” and he scribbles on the back of a napkin how you can set this up: you get a benevolent billionaire to set up a treatment system in a third-world country and start gathering data points, so that you can analyze the effectiveness of the treatments. What we’re trying to do with the compiled binary is reverse engineer the source code by data mining it against treatments. If you get that data set, then we’re there.

He’s like, “Yeah, but I’m not really a bio-informatics guy.” What’s he working on? Cat pictures. Now he was like a closet superhero. The dude sitting next to him was wearing a cape. I’m not making this up. All right. He wasn’t even trying to hide his superhero status. You guys are superheroes, and we’re all working on, well, crap. Most of us anyway. Now, if we started focusing on these problems, (I know it sounds kind of crazy) we would actually have a chance at solving them.

If we just focus on scaling, we’re going to scale up FarmVille. It’s going to be Farm Planet, and I’m not saying FarmVille doesn’t have value. What’s needed here is pretty obvious: it’s a culture change. We all need to be auteurs. We need to start buckling down now and preparing for getting old and getting interested in medicine and so on. Start studying our math. Start paying attention to data science. It’s not a specialty discipline anymore. It’s a generalist discipline. It should be.

Even at Google, where there are a lot of people who know a lot about data mining, it’s still kind of the haves and the have-nots, like me, who don’t know it. Well, O’Reilly, amazingly enough, seems to be actually trying to change the world with this convention and with Strata. I mean, they’re actually saying, “Hey, gosh, look data mining, machine learning, data analysis, math.” Well, okay, if we want a culture change, I've got a challenge. I've got a challenge for O'Reilly and a challenge for us. The challenge for O'Reilly is to publish a bunch of books on math for programmers, because O'Reilly's got a formula for making information accessible. Well, let's go out on a limb. Let's do physics. Let's do bio-informatics. Why not? Let's give them two years to publish these things. Then in that two year period, after they've published them all, we will agree to buy them and read them and prepare ourselves to solve important problems five years from now — two years from now.

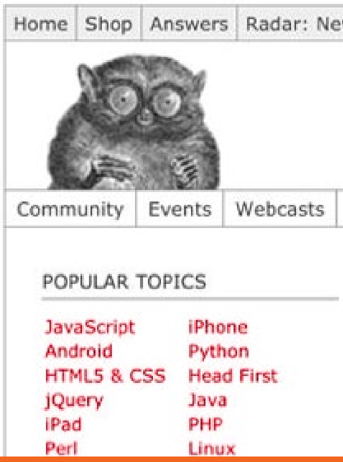
Whoa, I meant two seconds. They published them already. Bioinformatics, PERL Computer Skills, Mastering PERL for Bioinformatics: an O'Reilly bioinformatics book. A lot of these came out just in the last two or three years. Hey, I didn't even know they existed, but something's going on here. I'm kind of reading between the lines. I'm kind of making some educated guesses, but I know that these books aren't selling. So why are they publishing them? It's because Tim and his team are trying to change the world by affecting a culture change.

So the ball is totally in our court, and what's the ball doing? Well, the ball is apparently doing iPad. This is what's popular right now on O'Reilly's site. I just took this screen shot yesterday. Head First, that's kind of mathy.

Obviously, O'Reilly is principled here, because if they were just trying to make a buck, they'd be throwing OSCON iPad wouldn't they? They would, and they probably talked about it at length. But no, they're doing OSCON Data.

So, why don't we just effect a culture change here? Yeah, I'm not saying drop what we're doing right now. I mean, the world needs cat pictures. But what if we all started studying data mining and so on? And if you already know that stuff really well, then shame on you, because you're only one step away from solving hard problems, important problems, world-changing problems. You've just got to learn the domain knowledge. You don't need to learn it overnight.

I did compiler stuff, static program analysis at Google. I didn't know any of it four years ago. Nothing. I had a compiler class in school. Now, I'm not like Walter Bright or James Gosling, but I know compilers pretty well. I'm pretty proficient with them, and I can solve the problems that I need to solve. Same thing goes for this.



POPULAR TOPICS

JavaScript	iPhone
Android	Python
HTML5 & CSS	Head First
jQuery	Java
iPad	PHP
Perl	Linux

So, here's the funny thing. I had a mid-life crisis instantly after writing this. I went, "Oh God, I'm not following my own advice." So I dragged my math books down from upstairs and put them on the table here, and my wife and I are having study hour every day from now on.

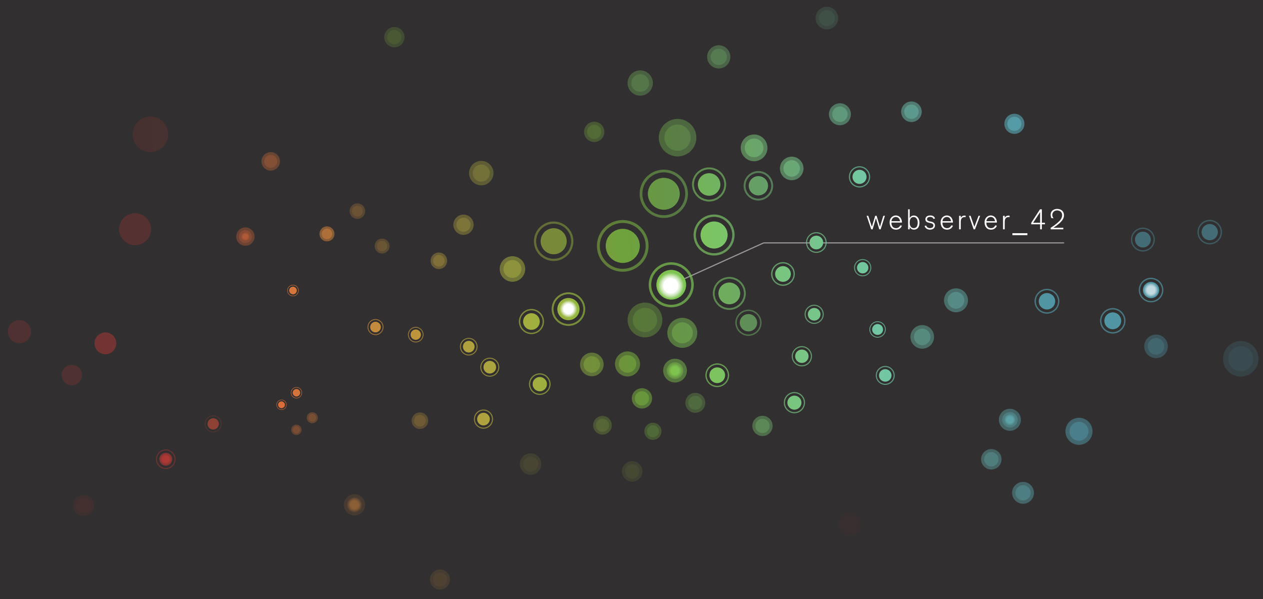
Moreover, this is part of my mid-life crisis, and this is going to come as a real shock to my boss. I had just signed up to work on a cat picture project. I told everyone I was going to do it, too, senior VPs and blah, blah, blah, blah. I'm sure you can guess what it was related to. I officially quit that job on national TV. So, you might say that I've got a little bit of skin in this game. I'm going to learn this stuff, so that five years from now — or three years from now, however long it takes — when the systems are scalable enough to handle the Human Genome Project (because they are, the thing practically fits in your pocket) and all of the other hard problems that we can solve with data mining, I will be ready. And I hope you are there with me. ■

Steve Yegge is a Staff Software Engineer at Google. Prior to Google he worked at Amazon.com as a Senior Software Development Manager. He earned his Computer Science degree from the University of Washington, and has over twenty years of experience as a software developer, dev manager, programmer hobbyist and tech blogger. Steve's current interests include language analysis, GNU Emacs and GNU Lilypond.

Published with permission of the original author. Originally presented at OSCON Data 2011 [hn.my/oscon].
Photo courtesy of O'Reilly Media and James Duncan Davidson.

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

Don’t Burn Bridges

A 4-Step Guide to Networking in Silicon Valley

By KAPIL KALE

NETWORKING IN SILICON Valley should be the easiest thing in the world. The Valley thrives on a free exchange of new ideas, so people always want to meet other people who might have interesting ideas or connections. As a result, people I hardly know introduce me to people they hardly know. Sometimes I get cold-emailed.

On general principle, and since the numbers aren’t currently overwhelming, I’ll take any introduction that comes my way. Most of the time, I’ll go out of my way to help those people out. Most founders, YCombinator or not, would do the same.

However, about half of people I’m introduced to do something that makes me not want to help them out. I’ve been through enough of these intro cycles now that I’ve started to discern four distinct patterns. Here’s what they are, and how to avoid them:

❶ Flaking on meetings

Last week, a kind-of acquaintance asked me for some advice on his fledgling startup, and wanted to see if I’d meet with him to discuss. I offered to have lunch with him the following week. That was the last I heard from him.

Radio-silence isn't the worst thing in the world. I'll still take the meeting in a couple weeks if the founder does end up getting back in touch. But I'll be far less likely to help that person, since he's going to have to really impress me to make up for my current impression that he's a flake.

❷ Forgetting to do follow-ups

One of my VC friends introduced me to a startup that he believed had a lot of potential. I took a 45 minute call with their main guy, who seemed full of energy and excited to get things done. I told him that they should apply to YCombinator, and offered to mention their application specifically to a partner to make sure it got a close look, under the condition that I could review it first. This is the type of thing I would have killed for before I was in YC.

Unfortunately, that founder never sent me his application.

A better networker would have sent me a draft the next day. Or even politely sent an email later that night that said "We decided not to apply. Thanks again for the feedback." But I won't ever recommend this founder for YC again.

❸ Being transactional

Earlier this year I was introduced to another founder working on a music startup. Their team needed help with their YC application, so I spent 2-3 hours helping them revise and improve it. After I sent them a last piece of feedback about their video, I never heard back from them again.

When I invest time in others, I like to hear how things go. In this case, I felt used. I have little interest in helping this founder out again. For us, we made sure that upon launch we sent out GiftRockets to people who helped us out.

❹ Not writing thank you notes

One startup founder cold-emailed me after our site launched, and asked me a couple questions. After I responded, he sent me a thank you email for my time. I later took an hour-long call with him to give startup advice. I'd be happy to introduce him to anyone I know. And I've never even met the guy.

I'm never peeved if someone doesn't write a thank you note, but I'm consistently impressed when someone does. Thank you notes after meetings are rare. At GiftRocket, we took a page out of the Wufoo book and started giving t-shirts to the people who helped us out. It's great for our brand, and it lets people know that we really appreciated their time.

Everyone in the Valley is busy, and I'm sure I've violated my own rules several times. But by just being a little bit thoughtful about your interactions with the people you meet in the startup scene, you can easily avoid burning what might be some very important bridges. ■

Kapil is a co-founder of GiftRocket [giftrocket.com], which was in the YCW11 class. He used to be a management consultant, and before that he went to Dartmouth College and got a degree in Economics.

Design Secrets for Engineers

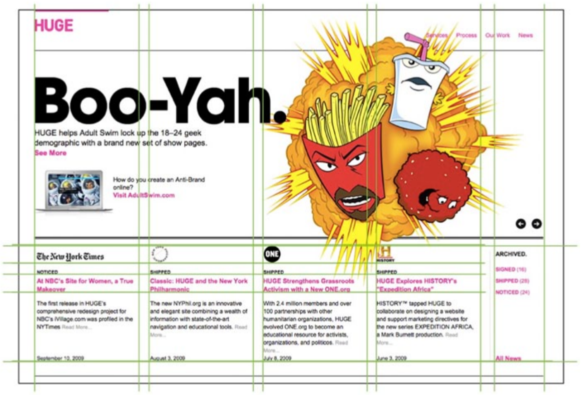
By PURIN PHANICHPHANT

If you are a designer like me, you must be asked on a regular basis to "make it look pretty." The request can stroke your designer ego, making you feel like a design rockstar with super powers to make this world a more beautiful place. This is especially true at startups, where you are one of the few, maybe the only designer there. However, it can also be really annoying — almost degrading at times. Thoughts like "why the hell can't engineers do this on their own? It's all common sense" always go through my head. If only engineers knew how to do visual design, designers would have more time to focus on cooler, more exciting problems like future product concepts.

And if you are an engineer, you might wonder how designers pull off their tricks (and why they're in such huge demand right now). Is it genetic? Do design schools teach them top secret design tips? Or did they make a deal with the devil to get designers' eyes in exchange for their souls?

Well, I'm here to bring you some good news: engineers don't need to drink unicorn blood just to be good at visual design. I am a strong believer that good design is a highly learnable skill, like riding a bike, playing a piano, or learning Spanish. If you practice often enough, you'll become better and better at it, and once you've got the hang of it, you'll never go back. I can say this because I, too, once sucked at design. But then I learned a few tips from my graphic design friends, and a few years later, I could proudly say that I was a design expert. Today, I want to share these not-so-secret tips with you. The first five are more specific to visual design while the next three are geared towards interaction design.

❶ Line things up.


HUGE
Boo-Yah.
HUGE helps Adult Swim lock up the 18–24 geek demographic with a brand new set of show pages.
See More
How do you create an Anti-Brand online?
Visit AdultSwim.com
The New York Times
ARCHIVED.
All News

Good example: beautiful sites and apps usually have underlying grid behind them.

This rule is the mother of all graphic design rules. Unless you're recreating the Mona Lisa on MS Paint, please line things up. Our brains just like it better that way. The slightly more advanced version of lining things up is called the grid system, which is essentially lining more things up. Kindergarten kids can do it and so can you.

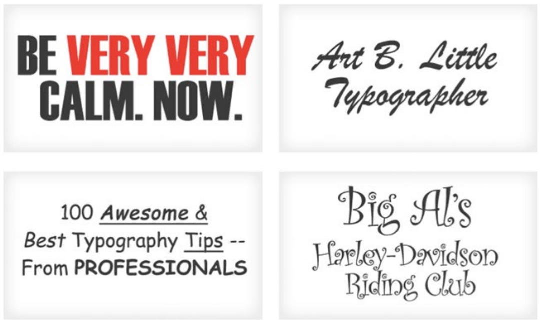
❷ Design the white space.

The Microsoft "Live" Platform
US-CERT
Design for Social Change
meltdown
Sun Dec 13th 8pm MILGIS
"Let's be efficient with space"

Bad example: this is what happens when you try to fill in your white space with information.

When you're in an elevator with 15 other people, it's not so easy to breathe…especially when someone farts. When you design a layout or UI, try not to jam too many elements into a page; it increases the chance of having one of the elements stink the whole thing up. Leave some white space for the eye to breathe. I often find myself designing the space in between elements, making sure elements aren't too far apart or too close together.

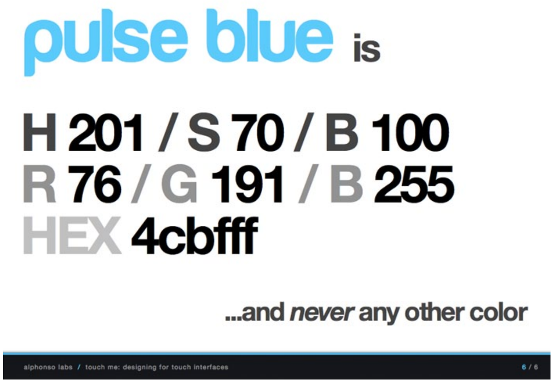
❸ Use designer fonts.


BE VERY VERY CALM. NOW.
Art B. Little Typographer
100 Awesome & Best Typography Tips -- From PROFESSIONALS
Big Al's Harley-Davidson Riding Club

Bad example: use these fonts and designers will make fun of you.

In the design world, there are good fonts and bad fonts. Good fonts like Gotham, Trade Gothic Bold Condensed or Garamond please your eyes and make you feel like you're having a frosty cold mojito on the beach. Bad fonts, on the other hand, make us designers cringe and feel like we've vomited from our eyes. Try to avoid super default fonts like Impact, Curlz, or Comic Sans, to name a few. If you must use a preloaded font, Helvetica and Georgia are two exceptions — they're classic and restrained enough to be inoffensive. If you want designer fonts that play nicely with the web, try Typekit. Oh…and please don't use WordArt. Ever.

❹ Keep it consistent.



pulse blue is

H 201 / S 70 / B 100
R 76 / G 191 / B 255
HEX 4cbfff

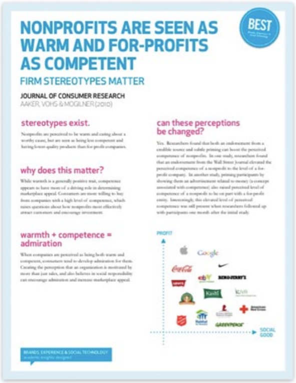
...and *never* any other color

alphonso labs / touch me: designing for touch interfaces 6 / 6

Good example: pick a few and run with it.

Use no more than two fonts and three colors in your designs. And keep them consistent throughout your sketches. Each time something changes, our brain has to go "whaaaa?" for a moment before figuring things out, so let's give our brains a rest and keep things consistent. Also, let's try not to stretch logos or images. Imagine if someone took your face and stretched it horizontally by 5%. Still happy with the way you look?

❺ Keep visual hierarchy in check.


NONPROFITS ARE SEEN AS WARM AND FOR-PROFITS AS COMPETENT

FIRM STEREOTYPES MATTER

JOURNAL OF CONSUMER RESEARCH
AAKER, VOHS & MOGILNER (2010)

stereotypes exist.

[illegible]

why does this matter?

[illegible]

warmth + competence = admiration

[illegible]

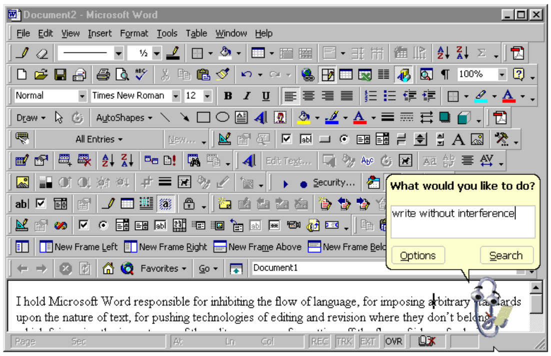
can these perceptions be changed?

[illegible]

Good example: squint your eyes. What do you see?

I don’t know about you, but when I cook, I always do a little tasting from time to time to make sure that my seasoning is on track. When you design, check yourself from time to time. Squint your eyes every now and then and look at the screen. What pops out at you first? What do you see second? Third? Walk away from the screen, and then look at it from 10 feet away. Believe it or not, designers and architects do this all the time to keep things in perspective (literally). It’s a good way to keep you from getting lost in little details or adding unnecessary buttons to the screen.

❻ Set priorities and stick to them.


What would you like to do?

write without interference

I hold Microsoft Word responsible for inhibiting the flow of language, for imposing arbitrary standards upon the nature of text, for pushing technologies of editing and revision where they don’t belong

Bad example: don’t allow this to happen in your product.

“Let’s put a help button there just in case the user is curious what’s going on. Oh…and let’s make the button look a little more like a button. And before I forget, can you make that tab pop a bit more?” Sometimes, I wish there was a robot that would bitchslap the one asshole in the room who keeps bringing up corner cases (cases that apply to only one very specific scenario). Until this bitchslap machine is built, however, we can get by with a list of what’s important and what isn’t, backed by some data if possible. It will save you time and energy, and shut that asshole up.

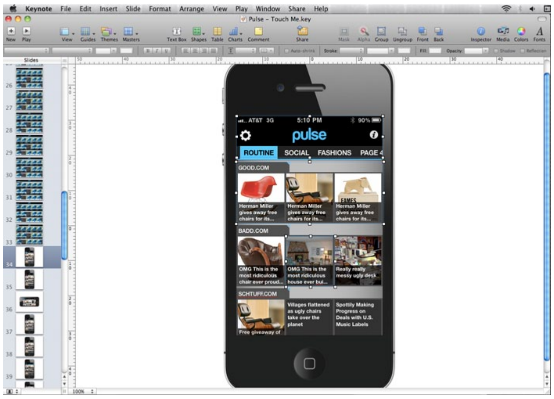
❼ Check the Physicality of the UI.

WTF example: imagine your interface lived in a little box. Try not to make impossible things happen.

A lot of what makes a UI successful is how familiar it seems to users when they first encounter it. Most users don't have exhaustive experience with mobile apps, and will assume that they follow the same rules as the real world. When you're making design decisions, ask yourself what sorts of physical analogs each element has. If the UI was to be re-created in the real world, would it make sense?

Though at different times, he's masqueraded as a Hollywood star, a part-time prince, and a buddhist monk, Purin has always been a designer at heart. He draws inspiration from his native Thailand, where fun, play, and lightheartedness are a way of life. Purin's fetish for mechanisms, buttons, knobs, and displays has led him to design a number of playful and interactive creations. And bikes.

❽ Use Keynote.



Best. Prototyping. Tool. Ever.

I love Keynote. I don't know how else I would have come this far in life without this magical program that lines things up automatically and makes it easy to make things look good. Beyond just making slide decks, Keynote is a great way to mock up UI flows. Do a quick web search for "Keynote mockup templates" and you'll find a number of great starting points for building good-looking prototype apps quickly and easily.

And remember to listen to others! It's natural that confidence comes with the thought that you're right and every one else is wrong. But just admit it: you're not always right.

Chances are you probably won't become a design supreme being over night. It takes some practice and confidence that you'll get good at it. Ira Glass from NPR sums it up pretty well [hn.my/ira].

Just keep repeating "I, too, can be a designer" — eventually you will become one. ■

Learn Vim Progressively

By YANN ESPOSITO

Want to learn Vim (the best text editor known to human kind) the fastest way possible. Start by learning the minimum to survive, and then slowly integrate all tricks.

Vim: the 6 Billion Dollar Editor

Better, Stronger, Faster.

Learn Vim and it will be your last text editor. There is no better text editor I know. It's hard to learn, but incredible to use.

I suggest you to learn it in 4 steps:

1. Survive
2. Feel comfortable
3. Feel Better, Stronger, Faster
4. Use Vim Superpowers

By the end of this journey, you'll become a Vim superstar.

But before we start, just a warning. Learning Vim will be painful at first. It will take time. It will be a lot like playing a musical instrument. Don't expect to be efficient with Vim in 3 days. In fact it will certainly take at least 2 weeks.

1st Level – Survive

1. Install Vim
2. Launch Vim
3. DO NOTHING! Read.

In a standard editor, typing on the keyboard is enough to write something and see it on the screen. Not this time. Vim is in *Normal* mode. Let's get in *Insert* mode. Type on the letter `i`.

You should feel a bit better. You can type letters like in a standard notepad. To get back in *Normal* mode just tap the `ESC` key.

You know how to switch between *Insert* and *Normal* mode. And now, the list of commands you can use in *Normal* mode to survive:

- `i` → Insert mode. Type `ESC` to return to Normal mode.
- `x` → Delete the char under the cursor
- `:wq` → Save and Quit (`:w` save, `:q` quit)
- `dd` → Delete (and copy) current line
- `p` → Paste

Recommended:

- `hjkl` (highly recommended but not mandatory) → basic cursor move (←↓↑→).
  Hint: `j` looks like a down arrow.
- `:help <command>` → Show help about `<command>`, you can start using `:help` without anything else.

Only 5 commands. This is very few to start. Once these commands start to become natural (may be after 1 full day), you should go on level 2.

But before, just a little remark on *Normal* mode. In standard editors, to copy you have to use the `Ctrl` key (`Ctrl-c` generally). In fact, when you press `Ctrl`, it is as if all your keys change meaning. With Vim in *Normal* mode, it is as if your `Ctrl` key is always being pressed.

A last word about notations:

- instead of writing `Ctrl-λ`, I'll write `<C-λ>`.
- commands starting with `:` must end with `<enter>`. For example, when I write `:q` , it means `:q<enter>`.

2nd Level – Feel Comfortable

You know the commands required for survival. It's time to learn a few more. I suggest:

Insert mode variations

- `a` → insert after the cursor
- `o` → insert a new line after the current one
- `O` → insert a new line before the current one
- `cw` → replace from the cursor to the end the word

Basic moves

- `0` → go to first column
- `^` → go to first non-blank character of the line
- `$` → go to the end of line
- `g_` → go to the last non-blank character of line
- `/pattern` → search for `pattern`

Copy/Paste

- `P` → paste before, remember `p` is paste after current position.
- `yy` → copy current line, easier but equivalent to `ddP`

Undo/Redo

- `u` → undo
- `<C-r>` → redo

Load/Save/Quit/Change File (Buffer)

- `:e <path/to/file>` → open
- `:w` → save
- `:saveas <path/to/file>` → save to `<path/to/file>`
- `:x` , `ZZ` or `:wq` → save and quit (`:x` only save if necessary)
- `:q!` → quit without saving, also `:qa!` even if there are some modified hidden buffers.
- `:bn` (resp. `:bp`) → show next (resp. previous) file (buffer)

Take the time to integrate all of these commands. Once you are done, you should be able to do everything you can do on other editors. But until then, it is a bit awkward. Follow me to the next level and you'll see why.

3rd Level – Better. Stronger. Faster.

Congratulation on getting this far! We can start the interesting stuff. At level 3, we'll only talk about commands which are compatible with the old vi.

Better

Let's look at how Vim could help you to repeat yourself:

1. `.` → (dot) will repeat the last command,

2. N<command> → will do the command N times.

Some examples, open a file and type:

- `2dd` → will delete 2 lines
- `3p` → will paste the text 3 times
- `100idesu [ESC]` → will write "desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu"
- `.` → Just after the last command will write again the 100 "desu."
- `3.` → Will write 3 "desu" (and not 300, how clever).

Stronger
Knowing how to move efficiently with Vim is very important. Don't skip this section.

1. N`G` → Go to line N

2. `gg` → shortcut for `1G`, go to the start of the file

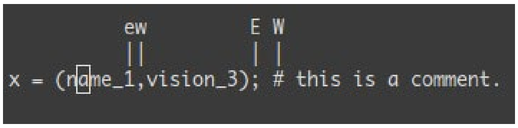
3. `G` → Go to last line

4. Word moves:

- `w` → go to the start of the following word,
- `e` → go to the end of this word.

By default, words are composed of a letter and an underscore character. Let's call a WORD a group of letter separated by blank characters. If you want to consider WORDS, then just use uppercases:

- `W` → go to the start of the following WORD,
- `E` → go to the end of this WORD.



```
          ew         E W
          ||         | |
x = (name_1,vision_3); # this is a comment.
```

Now let's talk about very efficient moves:

- `%` → Go to corresponding `(`, `{`, `[`.
- `*` (resp. `#`) → go to next (resp. previous) occurrence of the word under the cursor

Believe me, the last three commands are gold.

Faster
Remember about the importance of vi moves? Here is the reason. Most commands can be used with the following general format:

```
<start position><command><end position>
```

For example : `0y$` means

- `0` → go to the beginning of this line
- `y` → yank from here
- `$` → up to the end of this line

We also can do things like `ye`, yank from here to the end of the word. But also `y2/foo`, yank up to the second occurrence of "foo."

But what was true for `y` (yank), is also true for `d` (delete), `v` (visual select),`gU` (uppercase), `gu` (lowercase), etc…

4th Level – Vim Superpowers

With all the previous commands you should feel comfortable using Vim. But now, here are the killer features. Some of these features were the reason I started to use Vim.

Move on current line: `0` `^` `$` `g_` `f` `F` `t` `T` `,` `;`

- `0` → go to column 0
- `^` → go to first character on the line
- `$` → go to the last column
- `g_` → go to the last character on the line
- `fa` → go to next occurrence of the letter `a` on the line. `,` (resp. `;`) will seek for the next (resp. previous) occurrence.
- `t,` → go just before the character `,`.
- `3fa` → search the 3rd occurrence of `a` on this line.
- `F` and `T` → like `f` and `t` but backward.

A useful tip is: `dt"` → remove everything until the `"`.

Zone selection `<action>a<object>` **or** `<action>i<object>`

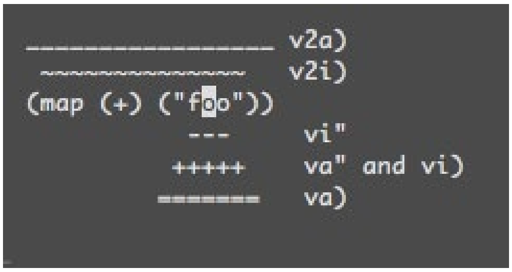
These commands can only be used in visual mode. But they are very powerful. Their main pattern is:

`<action>a<object>` and `<action>i<object>`

Where "action" can be any action, for example, `d` (delete), `y` (yank), `v` (select in visual mode). And "object" can be: `w` a word, `W` a WORD (extended word), `s` a sentence, `p` a paragraph, and also, natural character such as `"`, `'`, `)`, `}`, `]`.

Suppose the cursor is on the first `o` of `(map (+) ("foo"))`

- `vi"` → will select foo
- `va"` → will select `"foo"`
- `vi)` → will select `"foo"`
- `va)` → will select `("foo")`
- `v2i)` → will select `map (+) ("foo")`
- `v2a)` → will select `(map (+) ("foo"))`



```
----------------- v2a)
 ~~~~~~~~~~~~~~~  v2i)
(map (+) ("foo"))
           ---     vi"
          +++++    va" and vi)
         =======   va)
```

Select rectangular blocks: `<C-v>`
Rectangular blocks are very useful for commenting on many lines of code. Typically: `0<C-v><C-d>I-- [ESC]`

- `^` → go to start of the line
- `<C-v>` → Start block selection
- `<C-d>` → move down (could also be `jjj` or `%`, etc…)
- `I-- [ESC]` → write `--` to comment on each line

Note on windows you might have to use `<C-q>` instead of `<C-v>` if your clipboard is not empty.

Completion: <C-n> and <C-p>
In Insert mode, just type the start of a word, then type `<C-p>`, and then, magic...

Macros : `qa` do something `q`, `@a`, `@@`
`qa` records your actions in the *register* `a`. Then `@a` will replay the macro saved into the *register* `a` as if you typed it. `@@` is a shortcut to replay the last executed macro.

Example, on a line containing only the number 1, type this:

- `qaYp<C-a>q` →
  - `qa` start recording.
  - `Yp` duplicate this line.
  - `<C-a>` increment the number.
  - `q` stop recording.
- `@a` → write 2 under the 1
- `@@` → write 3 under the 2
- Now do `100@@` will create a list of increasing numbers until 103.

Visual selection: `v`, `V`, `<C-v>`
We saw an example with `<C-v>`. There is also `v` and `V`. Once the selection is made, you can:

- `J` → join all lines together.
- `<` (resp. `>`) → indent to the left (resp. to the right).
- `=` → auto indent

Add something at the end of all visually selected lines:

- `<C-v>`
- go to desired line (`jjj` or `<C-d>` or `/pattern` or `%` etc…)
- `$` go to the end of line
- `A` , write text, `ESC`

Splits: :split and vsplit.

Here are the main commands, but you should look at `:help split`.

- `:split` → create a split (`:vsplit` create a vertical split)
- `<C-w><dir>` : where dir is any of `hjkl` or ←↓↑→ to change split
- `<C-w>_` (resp. `<C-w>|`) : maximize size of split (resp. vertical split)
- `<C-w>+` (resp. `<C-w>-`) : Grow (resp. shrink) split

Conclusion

That was 90% of commands I use every day. I suggest you learn no more than one or two new command per day. After two to three weeks you'll start to feel the power of Vim in your hands.

Learning Vim is more a matter of training than plain memorization. Fortunately, Vim comes with some very good tools and an excellent manual. Run `vimtutor` until you are familiar with most basic commands. Also, you should read this page carefully: `:help usr_02.txt`.

Then, you will learn about `!`, folds, registers, the plug-ins, and many other features. Learn Vim like you'd learn piano and all should be fine. ■

Yann Esposito is the author of YPassword. He co-founded GridPocket and is an active web and iOS developer. He has a post Ph.D. in Machine Learning. He has written two research tools: dees & SEDiL.

Reprinted with permission of the original author.
First appeared in *hn.my/vimp* (yannesposito.com)

Coding Backwards

By JAMES O'BEIRNE

For a while I've wanted to write some personal finance software. Call me crazy, but I can't find anything currently in existence that allows me to project my earnings and expenses, then summarizes the net effect over a period of time. Basically, I want to be able to propose a budget to myself and simulate how it will play out.

So I sat down (as so many programmers have before me), ready to (almost certainly) reinvent the wheel, and I began to code. I wasn't really feeling inspired, though. A good design wasn't immediately clear to me, so I got slightly frustrated and stared aimlessly at the screen awhile.

Until I got an idea.

Quick Flashback

Last winter, I interviewed at a hedge fund called Two Sigma in New York City. Regrettably, I bombed the interview, but it was a great experience nonetheless. My first technical interviewer was a guy who looked like he needed no less than three pots of coffee to show any sign of human emotion. He asked me the boilerplate questions about design patterns and data structures, but afterwards he asked me to tackle a considerably difficult data-handling problem in Java. I cracked open a terminal-based session of vi as he shook his head and muttered something about why wasn't I using Eclipse.

Then, to my surprise, he told me to write out a bunch of class skeletons — class names, method signatures, and attributes. He wanted me to enumerate the cast of characters that I was going to be interacting with before I actually decided how the interactions would go down.

I walked out of the interview knowing I didn't get the gig, but I was intrigued by this guy's approach.

The Backwards Art of Software Design

I reminisced about the Two Sigma episode yesterday, as I was struggling to get down a design for my little personal finance Python API. I decided that I'd do the stoic Two Sigma interviewer one better and *write out a script using the yet unwritten API*. I'd reverse-engineer a good design by pretending I'd already written one!

Ten minutes and a few backspaces later, I came up with this:

```
from miser import *

m = Miser("jobeirne")

g = Goal(amount = 16e3, #$16,000
         by = Date(2012, 8, 1))
         # by Aug. 1, 2012

m.attachGoal(g)

bills = [Expense(name = "MATH315
tuition",        amount = 1.3e3,
                 on = Date(2011,
8, 29)),

         Expense(name = "netflix",
                 amount = 14.,
                 on = MonthlyRe-
curring(15))] # 15th day of the
month

income = [Income(name = "phase2",
                 amount = 1.5e3,
                 on = MonthlyRe-
curring(7, 22))]

m.attachExpenses(bills)
m.attachIncome(income)

print(m.summary(
      fromdt = Date(2011, 8, 20),
      todt = Date(2012, 9, 1)))
```

Skipping Steps

In a few ways, the design I came up with coding backwards was better than what I originally proposed. Initially, I was going to have classes like MonthlyIncome andDailyExpense that mandated an overly complicated inheritance structure to avoid repeating periodicity code.

When I wrote a script in my fictional API, I decoupled the transactions and their frequency of occurrence into two disparate objects without really thinking about it; instead of my initial design,

```
MonthlyExpense(name = "netflix",
               amount = 14.);
```

I now had

```
Expense(name = "netflix",
        amount = 14.,
      on = MonthlyRecurring(15))
```

which makes for a way more reasonable class tree (goodbye, multiple inheritances).

I was surprised at how satisfying the experiment was. A better design quickly appeared when I forced myself to preemptively eat my own dog food. Instead of shoehorning use-cases into a class structure I'd already designed, I coded backwards and the opposite happened: a design evolved from daydreaming about an API that I'd like using.

Paint by Numbers

Now that I had written out an interface I liked, development was simple: I started out with skeleton classes and then filled them out so as to match their behavior in the example usage I had written.

The process was straightforward, though I had to put some thought into how the classes would be structured internally. As I fleshed out the classes, I noticed a few ways I could improve upon the API I had come up with, so I made a few minor changes to the example usage. This iterative back-and-forth continued throughout development.

Snags

My experiment wasn't without a catch; since this method of coding backwards is about as top-down as you can get, I fell into the pitfall of doing a ton of development without actually running the code to test correctness.

This, of course, led to a long bout of debugging at the end, which was a slight pain, especially since I'm so used to rapid prototyping. Maybe if I had test-driven the filling out of the classes, things would've gone down smoother.

The project was so small that it wasn't a big deal, but I can see how my approach wouldn't have scaled to something much larger. I'm usually a big fan of bottom-up development, but coding backwards put me in a mindset that made that a little less straightforward.

Conclusion & Buzzword Bingo

Clearly, this technique is inspired by test-driven development, and for all I know it may be old news to experienced programmers. Nonetheless, I found using it to be a fun experiment and a nice addition to my utility belt.

There is no silver bullet for development methodology. Coding backwards, or interface-driven development, certainly won't solve all your problems. It is, though, a quick and gratifying way to help writer's block and gain some clarity when you sit down to write an API. ■

James O'Beirne is a web developer living in Alexandria, VA. He works at Phase2 Technology, where he crafts robust web applications for clients in publishing and government sectors. Previously, James worked at the National Institute of Standards and Technology developing an open-source scientific computing framework, FiPy.

Reprinted with permission of the original author.
First appeared in *hn.my/backwards* (jameso.be)

Competitive Game Design: The Marginal Advantage

By SEAN PLOTT

I RECENTLY WAS INVOLVED in a Mancala competition, where the entrants had to code an artificial intelligence program that could play Mancala. It taught me an important lesson about competitive game design.

Mancala is a game in which the winner is the player who "captures" more stones than his opponent. Thus the winner of the competition was the entrant whose AI program bested all others in stone capture. One of the coders devised a computer program that would maximize the number of stones captured in a given turn. Throughout the tournament, he steamrolled his opponents, repeatedly winning by fifteen stone margins and instilling fear and despair in the hearts of other coders. However, his program ultimately lost in the finals by 0-2, and lost each of those games by exactly one stone. In fact, I was shocked to hear that the winning program had consistently won every game it played by exactly one stone. How could the first program, which seemed to terrorize opponents, lose to another program that could only barely squeak out a victory each time?

The results were explained by subtle differences in their approaches to game play. The first player wrote a greedy program, one that would gobble up as many stones as possible. The coder reasonably theorized that maximizing the number of stones would maximize his chances of winning. However, the winning coder displayed even greater insight into the game: his goal was to have more stones after both players had taken a turn. In a sense, after taking the lead, he chose to maintain that lead, rather than extend it. For example, instead of capturing ten stones in a turn, he would capture four, knowing all the while that his opponent could only capture three, and that his lead would thus be extended by one. In a sense, the first programmer's AI was successful in maximizing the number of stones captured in each turn, it just happened that this was always one less than the winner.

single commonality: they play comfortably with a marginal advantage.

The marginal advantage embodies the notion that one cannot, and should not, try to "win big." In a competitive setting, the strong player knows that his best opponents are unlikely to make many exploitable mistakes. As a result, the strong player knows that he must be content to play with just the slightest edge, an edge which is the equivalent to the marginal advantage. More importantly, a one-sided match ultimately carries as much weight as an epic struggle.

I found this incident to be particularly intriguing, as it reflects the nature of successful competitive game design. I've been involved in the competitive gaming community since 2001. Although my primary game is StarCraft, I have considerable experience with WarCraft 3, CounterStrike, Marvel vs Capcom 2, and a variety of other games. Despite the fact that these games function in drastically different ways and demand completely different skill sets, the expert players, the players who consistently win, always share a

After all, the match results only in a win or a loss; there are no "degrees" of winning. Therefore, at any given point in a game, the player must focus on making decisions that minimize his probability of losing the advantage, rather than on decisions that maximize his probability of gaining a greater advantage. In short, it is much more important to the expert player to not lose than it is to win big. Consequently, a regular winner plays to extend his lead in a very gradual, but very consistent manner.

Amateur players, on the other hand, try risky, greedy strategies. In CounterStrike, for example, it is not unusual for amateurs to dash out into crowds of enemies trying to pull off a miraculous string of headshots in order to eliminate the opposing team. The majority of the time, this kind of amateur is fragged in a nanosecond. Expert Counterstrike players, on the other hand, patiently and carefully pick off enemies, knowing that such caution and precision virtually guarantee a win.

So, what's so special about the marginal advantage? It might seem that all I've done is imply that newbie players take unnecessary risks and experts do not. However, just as playing for a marginal advantage is the hallmark of the expert player, the presence of a potential marginal advantage in a game is the hallmark of excellent competitive game design. The Mancala tournament brought this sharply into focus for me. Building in and allowing for a marginal advantage leads to exciting and dynamic play.

"Building in and allowing for a marginal advantage leads to exciting and dynamic play."

Moreover, amateurs often have no idea what to do with a marginal advantage once they gain one. I have personally watched countless games of StarCraft in which a player gained a massive lead but later lost the game. An opponent moves out a large force, and the amateur annihilates it with ease. At this point, the amateur has a marginal advantage: he has not yet won, but his opponent has lost his military and cannot apply any pressure for some time. The amateur in this situation will immediately try to win by launching a counterattack and will then crumble to a strong defense. Alternatively, the amateur will expand excessively, over-extending his bases to the point where his defenses are too thinly spread. Such decisions violate the law of the marginal advantage, as they allow the opponent to get back into the game. They erroneously attempt to extend the lead, as opposed to maintaining it.

Not that providing for a marginal advantage is the only critical element in competitive game design. I will concede that all reasonably designed competitive games share three basic traits: ambiguity of optimal play, diversity of play, and allowance for skill.

First, there is no such thing as a good competitive game that has an obvious optimal strategy. For example, there is no competitive tic-tac-toe gaming, as two logical players would tie every time. I remember playing an old real-time strategy game called KKND, where one of the units was clearly the strongest unit in the game. Playing against my brother degenerated into a fray where we hurled masses of the same unit at each other until one of us got bored and stopped playing. On the other hand, a superior game like StarCraft presents a completely intractable problem. There is no best race, no best strategy, and certainly no best way to win.

Second, a quality competitive game should have a variety of techniques that can be employed in order to win. That is, if a player performs strategy A, the opponent should have more than one reasonable response as an option. Games frequently have a system of built-in counters. Theoretically, game creators insert such counters to avoid the danger of an optimal strategy. However, these kinds of systems often cause games to be nothing more than a fancy multimedia version of rock-paper-scissors: Unit A counters Unit B counters Unit C counters Unit A. On the other hand, excellent competitive games, such as Marvel vs Capcom 2, allow for huge diversity in response. In Marvel vs Capcom 2, players can elect three characters to form a fighting team. With fifty-six characters from which to choose, Marvel vs Capcom 2 offers over 25,000 combinations of possible teams, presenting the player with virtually unlimited options. At the highest level of play, strong competitive players can be seen using drastically different teams and styles.

Third, a good competitive game should test a player's skills and minimize the element of chance or luck. Ideally, the probability of a weak player defeating a good player should be as close to zero as possible. For example, in a well-designed game like WarCraft 3, it is highly unlikely that an amateur will be able to control his units or respond to his opponent's tech patterns as well as an experienced player. In fact, the best way to test a player's skill in a game is to present the player with more decisions. In WarCraft 3, a player not only has to make major decisions, such as which buildings to make or what hero to choose, but also has to make innumerable small decisions, such as how to precisely control each unit or time an attack. By presenting a player with more decisions, the game offers amateurs more opportunities to make mistakes and experts more opportunities to shine.

However, against this framework of competitive game design, we can understand why the marginal advantage gives a game flavor and excitement for both the player and observer. The marginal advantage not only provides the player with the joy of overcoming obstacles, of finding new and more effective methods of winning, but also allows a player to express himself, to have his own unique style. By exploiting the marginal advantage, the expert player is both a problem solver and an artist. In WarCraft 3, StarCraft, Marvel vs Capcom2, and CounterStrike alike, we see the individuality of the players shine through: some play aggressively, some play defensively; some are renowned for their solid, steady play, others for their unorthodox tactics. We respect the brilliance of their expert skills; we admire their ability to win. Yet, at the same time, we appreciate the aesthetic of each player's technique, that each player finds a solution that is so different from the next player. If too many decisions are clear cut, the player has no need to discover his own marginal advantage over the field, and the competitive game collapses into redundant, unexciting play, unappealing to master and unappealing to watch. The brilliance of a competitive game is that the designer must limit his role to be the creator and balancer, to allow for the potential of innovation. In this way, each player can uncover his own marginal advantage and become the true pioneer of the game.

Whenever I encounter some little hitch, or some of my orbs get out of orbit, nothing pleases me so much as to make the crooked straight and crush down uneven places. ■

Sean Plott, commonly known as Day[9], is a former professional Starcraft player and a sportscaster known for his Day[9] Daily Netcasts at *day9.tv*

This is your website.

This is your website on Optimizely.

Learn more and save 20% by visiting:
optimizely.com/hackermonthlyB

Optimizely
Website A/B testing you'll actually use

Dream. Design. Print.

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

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

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

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

25% Off the First Issue You Publish

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

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

MagCloud