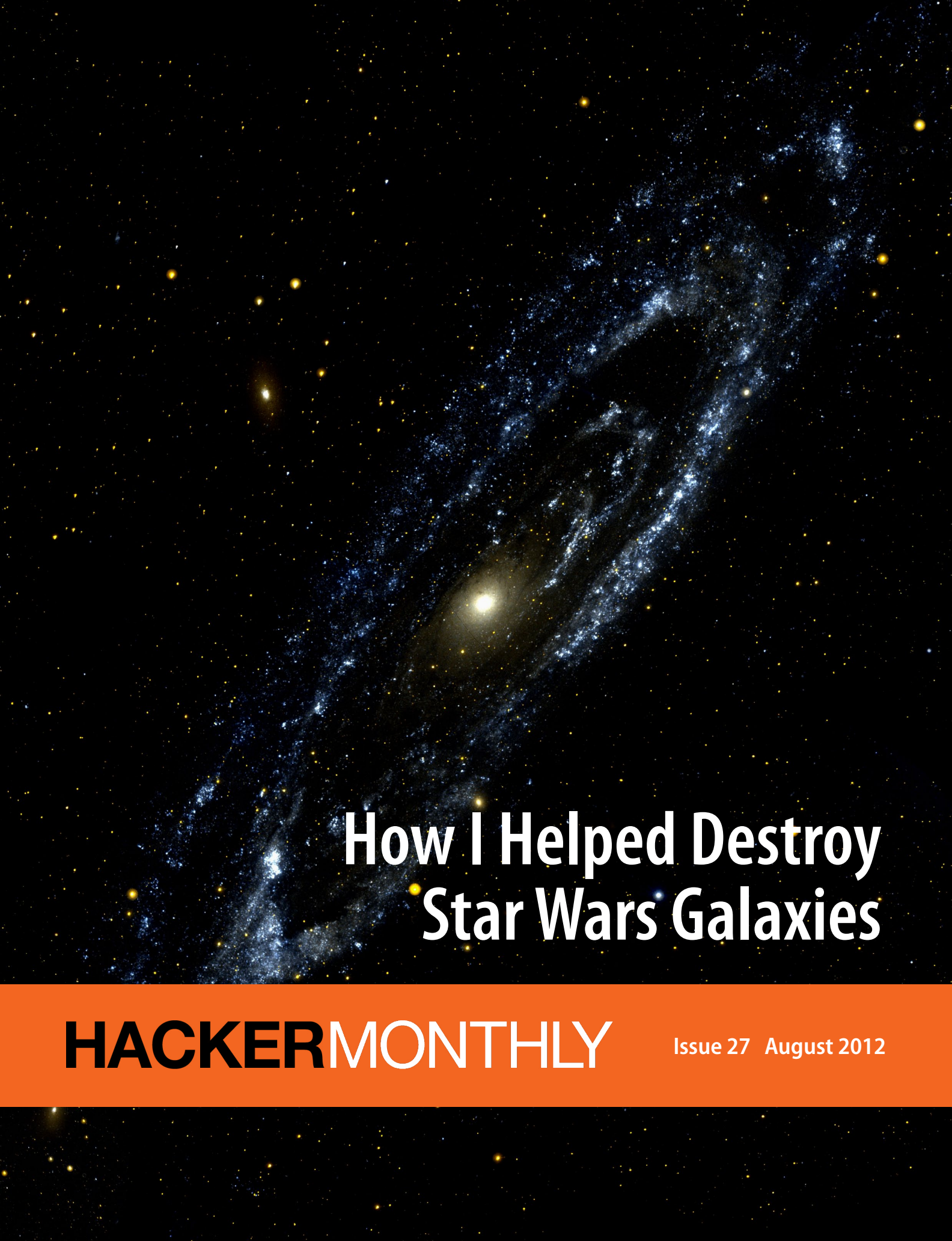


How I Helped Destroy Star Wars Galaxies

HACKERMONTHLY

Issue 27 August 2012

Curator
Lim Cheng Soon

Contributors
Daniel Tenner
Patrick Desjardins
Nate Kontny
Matthew Wensing
Sacha Greif
Jason Winder
Tom Kleinpeter
Matt Might
Reginald Braithwaite
Larry Osterman
David Peter
Dann Berg

Proofreaders
Emily Griffin
Sigmarie Soto

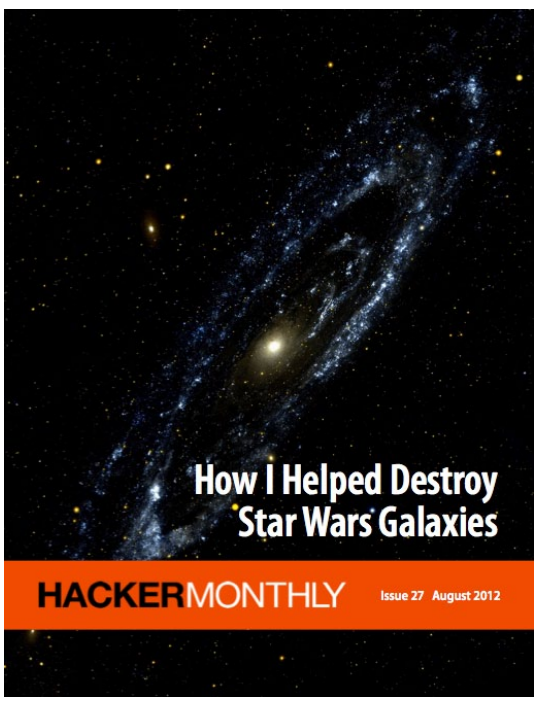
Printer
MagCloud

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

Advertising
ads@hackermonthly.com

Contact
contact@hackermonthly.com

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



Contents

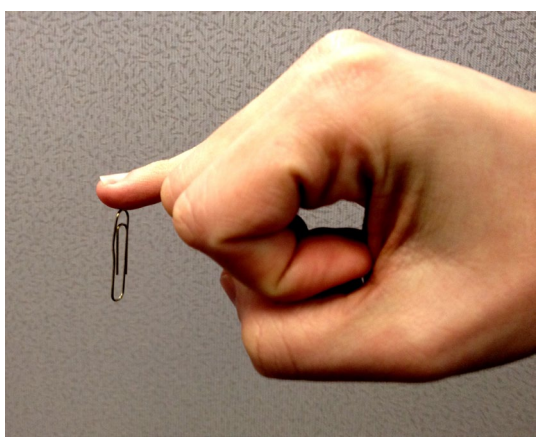
Illustration by Jaime G. Wong

FEATURES

STARTUPS

SPECIAL

PROGRAMMING



For links to Hacker News dicussions, visit *hackermonthly.com/issue-27*

The Salesman and the Developer

By DANIEL TENNER

A SALESMAN AND A developer go on a bear hunting trip. They arrive at the cabin in the woods and start unpacking the car, moving stuff into the cabin, and getting things ready for a week of bear hunting in the wilderness. The salesman quickly gets bored of this and says, "Tell you what, you continue unpacking and getting everything ready, and I'm going to go find us a bear."

The developer sighs and nods (he's used to salesmen) and continues setting up while the salesman vanishes into the woods.

Half an hour later, as the developer is about three quarters done getting things ready (the cabin is now all neat and tidy at last), he hears a very loud growl as he comes out of the cabin. Twenty meters away, the bushes start shaking. Out shoots the salesman. Right behind him, a huge, snarling, drooling, roaring monster of a bear. It's twice the size of a normal bear, and it's very, very angry.

> “Sales don’t happen without someone energetically pushing the product.”

As the developer hides behind a chair, the salesman runs right up to the cabin with the bear on his heels, and just as he’s about to go through the door, he quickly leaps to the side. The bear crashes past him right into the cabin, and the salesman deftly closes the door right it, locking the bear in. Loud noises can be heard as the bear begins trashing the inside of the cabin.

The developer emerges from behind the chair. The salesman cheers and says, “Woohoo! That’s the first one. Now, you kill him and skin him, I’ll go find us another!”

Two perspectives

There are two ways to understand this story, and which way you favored largely depends on whether you’re a “builder” type or a “sales” type.

If you’re a builder type, you see this as a great story that illustrates a common problem with salespeople: they don’t seem to care about what happens after they make the sale. Actually delivering the project is hard work, but by then the sales guys have moved on to something else, so they don’t care (and, as an additional problem in some industries, the salespeople will sell stuff that can’t be realistically delivered).

However, if you’re a sales type (like my cofounder, Paulina), you have a different perspective on this story. It’s yet another story that makes fun of salespeople while completely discounting just how hard it is to not only find that damn bear, but bring it back and get it through the door.

Who’s right, then? Both, of course. In business, you need both to find and sell clients, and the ability to then deliver what you sold them. One without the other is not a business.

Sales is not optional

Many people who “do startups” these days are from a technology background. In other words, they’re builders rather than salespeople. And, like all builders, they tend to disregard sales as something that can happen later, something secondary that we’ll solve when we get to it.

Well, sales isn’t secondary. Speaking as a builder type myself, and having experienced businesses both with competent sales and without, I now believe that having someone who can find clients willing to give you money from day one is so important that I would not start any company without such a person.

Sales don’t happen without someone energetically pushing the product, service, or whatever it is you’re intending to sell. Some may dream of products that sell themselves, like Dropbox or the original Apple II, but even awesome products like those took serious sales effort to get off the ground. Apple had Steve Jobs, one of the master salesmen of his generation, pushing the product everywhere he could and striking bold deals to get the company off the ground. Dropbox endlessly tweaked their referral scheme before they went viral.

A few businesses, like Google, Facebook, and Instagram, get to figure out the business model later. They can do without sales, perhaps. But this model only works in one place in the world, and unless you’re starting up in the Silicon Valley bubble, your business is not a business without sales. ■

Daniel Tenner is the founder of Woobius and GrantTree. Known as “swombat” on Hacker News and Twitter, he is now producing *swombat.com*, a daily updated resource for people who like to read startup articles like this one.

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

Illustration by Jaime G. Wong

How I Helped Destroy Star Wars Galaxies

By PATRICK DESJARDINS

I SAT IN FRONT of my laptop at work, watching the videos from the previous night. While logically I knew this was Star Wars Galaxies, I recognized nothing on the screen. It was like watching a completely different game. In that video, I saw the end to what could have been an amazing game, and I saw it end with a whimper. It was like a bloated corpse, already long dead and unaware of it. It was depressing.

In summer 2001, I started reading up on the upcoming game. It sounded awesome. We were still a long way from public betas, but I took a real interest in the online community that had already formed. We talked constantly, speculated, made suggestions, argued about how Jedi should work; we were two years from ever even playing and we already had deep and powerful opinions about a game that didn't exist yet. It was unprecedented. Many of us had already played EQ or UO. We knew what we wanted. We all had a deep love for the source material. We fantasized about force lightning and saber throws. We wanted to fly the Kessel Run with Han Solo and Chewie. We imagined arguing bounties with Jabba, fighting Darth Vader. We wanted it all, and Sony knew it.

I was 21 and had just sold my first business. Flush with cash and ready for my next adventure, I had no idea what it would be.

Spring 2002: The first sandbox alpha builds were being tested. Over the course of spring and summer, they got a little more advanced, and I could see the game starting to take shape. I got into the friends and family alpha tests from my involvement in the online community. I made copious suggestions, everything from combat to social aspects. I complained for a week about how the zabrak horns should look. I got involved, deep.

One day, I inquired as to how the economy would be structured. The answer I got very literally changed my life.

"We haven't really planned for much of anything. I think the players will structure it organically."

I was dumbstruck. I didn't respond and started taking notes. I took a lot of notes — entire composition books sat next to my monitor. In hindsight, 90% of what I noted was useless, but that 10% — that was worth something.

Early 2003: Beta is in full swing. We got our first real look at how things were going to work, and I saw the opening. The giant hole that no one in development saw, or cared about if they did. More so than anything else, this game would be about real estate and ease of use for crafters. Supply a convenient place for everyone to go and they will go there, even if that means paying a premium.

I spent a lot of time in starports, counting players arriving and leaving and establishing traffic patterns. Corillia, Naboo, Tattooine — the big three. I started running projections: where would I go first? Tat. Surely Tat, but…where would I want to live? Not in the desert. No way. Naboo, lush and green, pretty scenery, Fambas walking in the distance. Yes, this is where I would live. But there is also Coronet, the central hub for travel. If you want to go anywhere, you have to go through Coronet. That's the meeting place, the staging point; Coronet would be the key to power. If I wanted to hold the cards, I needed to hold Coronet. I started looking at the most efficient way to place buildings outside the c-net starport. I was placing them for hours, plotting the perfect placement to not only have the closest buildings, but also to force other players to build elsewhere.

I started thinking a lot about human nature. I started thinking about exploiting laziness and sloth. I started thinking of this game as a business model and less as a hobby. This was now something to be mastered and exploited. I scoured forum posts for shortcuts, for exploits, for bugs that would most likely make it through to release.

I started thinking about the crafting and the shortcuts there. I created timetables based on the initial samples we had. How many hours to master this, how many hours to master that? How many supplement accounts would I need to supply myself? If there are only 24 hours in a day, how could I best utilize each one?

I started building extra computers. I spent every spare moment preparing for day one.

On release day I was at EB games, cash in hand for eight copies of SWG, and I was home in a flash. I took a week's vacation. I had the spare bedroom stocked with food and drink, my computers arrayed in a half moon.

Those first two weeks are a blur. I don't remember details; I just remember the accomplishments. I remember when I mastered the first handful of professions. I remember screaming in frustration when my math wouldn't work due to slight changes in crafting between beta and release. I remember my wife growing increasingly concerned.

Slowly, steadily the credits started building. I kept a tally on a whiteboard leaned against the wall. Your first million is the hardest, they say. Bullshit. Your first 100,000 is the hardest. But I kept working, kept pulling 12, 14, 18 hour shifts in front of keyboards and tiny screens.

Little by little, my plan came together. Mistakes buried under accomplishments. Vendors multiplying like rabbits. Small houses, big houses, entire malls and cantinas. Credits piling up, stacks on stacks. Professions mastered, exploited, and dropped to master new ones.

I spent more credits in a day than most people would all year. At first my "competition" didn't get it, but I paid and they didn't care.

They were standing on a track and couldn't see the train. I wasn't slowing down. If anything, I just went faster.

I clearly remember the day that I realized I had done it. It was maybe two or three months in, and I controlled not only the land around Coronet, but Theed as well. It was mine: People used my vendors because they were closer, and for no other reason. Slowly I increased my prices, 2%, 5%, 10%… and they lined up to buy. People were holo-grinding and didn't care what it cost. It was a full-time gig just keeping the vendors supplied.

Six months in and I realized I had more money than I could ever possibly spend. I needed to off-load it, and I needed help. Enter the Thai.

His name was Tan, and he needed a reliable stream of credits. See, Tan worked for a re-seller and my little enterprise was making his job difficult. He had no problem on other servers, but on those that I was on, his percentages were way down.

Why not work together? Why not indeed. After a week of negotiations and arrangements we were set and money was changing hands with an interesting side-effect.

The same people who were buying my credits from Tan were turning around and using them at my vendors, usually with more of their own credits as well.

I was now making real-world money for making virtual money by making real money. It was amazing, and it worked perfectly. I would transfer 10 million credits to Tan; he would pay me via bank transfer. He would then sell the fake money for real money at around a 100% mark-up. The player would get his 500,000 or million and turn around and buy my merchandise for 1.5 million. This happened across the board, at all levels.

I remember with crystal clarity when I realized I was making more money from this enterprise than I was at my full-time job. I quickly decided to expand and hired four guys in Singapore to play 24/7. I paid them unreasonably well for the time, almost 3x as much as they would for other re-sellers; this bought me loyalty, and in this enterprise, loyalty is everything.

Soon the money was stacking fast and I needed to expand again, and again. At the peak, I employed 12 men and women. I controlled, for the most part, the economy on four servers, and I was bringing in almost a six-figure salary.

My wife went from hating the fact that I was obsessed with the game to helping me run the books and check on the numbers. She made suggestions on rates and agreements, and in turn, I bought her a car and we bought a house.

After almost two years, I could see that this would not last. Player counts were dropping; the game was being mishandled more and more. When they did away with the holo-grinding, it wrecked a large part of my business model. And again, when the Jedi-village went live, it was the final nail. No one needed to spend vast amounts on anything anymore. You could just become a Jedi from a quest chain.

I started shutting down my enterprise. I had bought and sold dozens and dozens of accounts, billions of credits; for the remaining players on my servers, my accounts were fixtures. They were how they functioned, they were how they survived. Most had no clue it was one person pulling all these strings, and in the end, I liked it that way. I stopped "playing" the day I was killed in Theed starport by a fresh new Jedi who didn't understand how to even play the game.

I couldn't even bring myself to fight back. I just stood there. I was one of the few true Dark Jedi Masters, and I let him kill me. That very act illustrated perfectly what SOE did wrong. Those of us who had faithfully put in the hours and weeks and months required to earn those lightsabers were spit on and betrayed by the very architects of the game we loved.

Now obviously I did my share of exploiting the game, and your share, and his, and hers. But I put in the work to holo-grind. I put in the work to move my way up endlessly grinding on fambas in Naboo, cats in Corrilea, and rancors on Dathomir. I didn't buy my personal Jedis; I earned them. I knew the game, I knew the struggle, and I knew what it took to get them.

And in the end? On my last day playing? You could start a new toon who was already a Jedi. I walked away and I never looked back. That moment at my desk, 10 years after it started, I sadly closed the window and went back to work.

Because it wasn't the game I loved. That game died in 2005 with the NGE/CU. It died when developers turned their backs on the gamers who had spent the effort and instead listened to the lazy, whining voices who wanted it all given to them.

Ironically, those voices were the same people who happily handed Tan money for the credits I provided. Happily handed me stacks of cash for Jedi accounts. Did I help in the demise of SWG? Yes. That is something I accepted long ago. The game that I loved so much, I helped to destroy. ■

Patrick Desjardins is an American programmer and graphic artist.

Reprinted with permission. First appeared in Medium Difficulty: *hn.my/starwars* (mediumdifficulty.com)

Founded by Karl Parakenings, Dinosaur McDowall, Nico Dicecco, and Kyle Carpenter in late 2011, Medium Difficulty is supported by a staff of a diverse variety of writers, all with an interest in critical analysis of games and their place in a larger cultural context.

Now you can hack on DuckDuckGo

DuckDuckHack

Create instant answer plugins for DuckDuckGo

duckduckhack.com

I Have No Idea What I'm Doing

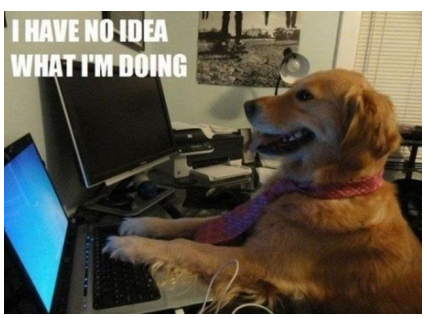
By NATE KONTNY

> **"Everything will be okay in the end. If it's not okay, it's not the end."**
>
> **— Somebody pretty wise**

Justin Kan recently posed the question: What good is experience? [hn.my/exp]

The ultimate good that comes from experience is that it teaches you this:

You'll constantly find yourself in situations where you have no experience and you have absolutely no idea what you're doing.



But here's the thing. You don't need the experience. You just need some grit.

Grit: courage and resolve; strength of character.

In other words, you can figure it out.

See, no matter how much experience I get, I continuously find myself in situations where I have no idea what I'm doing. I have countless personal tales of being neck deep in some type of problem or subject and being completely baffled about how I'm going to figure it out.

There was my freshman year honor's Algebra class. Before the first day I wondered if someone made a mistake placing me in a class like this. I mean, I was a pretty good student, but I didn't even have a decent pre-algebra class to prepare me. My suspicions were further confirmed on the first day.

The entire class was able to yell out the answer to this question: Expand the following expression:

*(1 + 2x) * (3 + 4x)*

Hold on, what!?

The class yells out, "FOIL!" and chants, "First…Outer…Inner…Last…"

I had no idea what I was doing.

After many nights of my mom trying to teach herself and me these types of problems, I was still totally f*&^ing baffled. It came to the point where I approached my Algebra teacher and fessed up that I'd only be a bit more lost if he was teaching the class in Italian and that maybe I should change classes.

Thank god for that Algebra teacher. I will always remember Mr. James G. Serpe. As he sucked on some Luden's cough drops (that man was addicted to Luden's cough drops), he told me it was fine to move to a lower level math class, if that's what I really wanted, but he warned, "I think you can figure this out."

Mr. Serpe had enough experience to have seen enough folks just as lost as I was eventually figure this stuff out. All it takes is some grit. Mr. Serpe said I could come over to his office hours after school.

I went every single day.

Eventually I figured the crap out of Algebra.

So much so, that I had the highest grade point average of my entire freshman class (over 400 students) at the end of my freshman year.

But the thing is I continuously found myself drowning in the subjects in high school. In college, it just got worse of course. And then after college? Oh my God. :)

There was the case of getting my first job after college. I wanted to be a software developer, but I got stuck doing really terrible stuff. My company didn't think I had the education to be a developer, so I basically had to collect things people talked about in meetings that represented "requirements" and stick them into Word documents. I hated it. I wanted to be a developer, but…

I had no idea what I was doing.

So I would stay after work, sucking down information from the internet on how to create websites and program software until the building would shut its lights off. Actually, I stayed even after that.

I downloaded Java so I could install it at home.

It took about 12 floppy disks. :)

In the end, I finagled my way into a new career as a software developer at that same company. I became a pretty good Java developer and then moved on to a senior engineer role at my next job. Then, I went on to be the technical cofounder of my own business, Inkling [inklingmarkets.com], in the second class of Y Combinator, where again, I found out…

I had no idea what I was doing.

There was the day when we realized the simple algorithm I was using to "score" people's use of our tool wasn't working. That "score" was the fundamental point of our business. And now, a guy had figured out how to completely game our system. My original algorithm was incredibly naive of course. I had never created something like that before, so I dug even further into the literature than ever before, finding algorithms from the best minds in the industry we were in (prediction markets).

I was looking at papers like these [hanson.gmu.edu/mktscore.pdf] from Robin Hanson, a pioneer in prediction markets. And I thought algebra was tough? Jesus, was I completely lost.

But I worked at turning Robin's equations and papers into something I could understand. Something I could put into Ruby code to fuel our software.

After weeks and weeks and weeks of battle, including a vacation weekend where my wife and I went camping, I just sat with a calculator and a notebook on our campsite looking to make all these algorithms and choices make some kind of sense.

I finally figured this thing out.

Then there was the time CNN was a client and stuck a link to Inkling on CNN.com for multiple days in a row. I had never experienced creating a dynamic website that could handle millions of people every single day (let alone using Ruby on Rails to do this).

I had no idea what I was doing.

But of course, I figured something out. I figured out the bottlenecks and performance problems to keep the few servers we had from melting down.

Fast forward to today. I've had over 6 years of experience helping to start and run my own profitable software business. Over those years, I've wanted to give up a countless number of times. These were just a few stories of dozens where I couldn't possibly fathom how we were going to figure out how to get through our next challenge. But we did.

Now I find myself in the middle of starting a new project and a new business. And…

I have no idea what I'm doing.

My first attempts at products in this new business haven't quite gone right. I'm working on the next "something" that I have some hope might be it, but I'm baffled at how it's going to work out and I have some insane competition in what I want to achieve. How am I possibly going to get through this?

But if experience has taught me anything…

I'll figure this out.

And you will too. ■

Nathan Kontny is the co-founder of two companies that have graduated from Y Combinator: Inkling and Cityposh. He likes to share stuff that inspires him at *ninjasandrobots.com*

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

The Anatomy of Profitable Freemium

By MATTHEW WENSING

Choosing the freemium business model can be either brilliant or deadly. The difference lies not only in the execution of the marketing, but also in the nature of your product and the design of your business. Can you coordinate all three to become sustainably freemium by making a profit?

Summary

- Freemium requires a business to maintain two value propositions with a well-designed interaction between the two.
- The premium offering can deliver different kinds of value. You should choose the most profitable one you can under your constraints, which best fits the value your product delivers.
- Using a free car analogy, the kinds of premium value (types of freemium) include: Cargo Freemium, Airbag Freemium, Bomb Freemium, Cruise Control Freemium, and Bells & Whistles Freemium.
- Each type of freemium has unique risks as well as market and funding requirements to be sustainably profitable.
- Any of these types of freemium can be poorly implemented, and some types may not work with your product. Correct implementation requires a deep understanding of your product's value.
- Entrepreneurs should be convinced that freemium is the right choice for their product/market, choose their type of freemium out of knowledge (not fear or ignorance), and re-evaluate this decision regularly.
- Freemium can be extremely beneficial for gaining market share, but if it can't create sufficient conversions to a premium offering, it isn't sustainable.

Not One, but Two

Your value proposition is the proposal you make to a prospective user or buyer, wherein they exchange their time or resources for the life-changing experiences (for better and for worse) your product provides. The freemium business model actually contains two such proposals — one set of experiences the user can get for free and another set of experiences that can only be acquired through purchase. To be successful (profitable), we need to understand when and in what manner (constructive or destructive) these propositions touch. In familiar terms: how, when, and why does the user "convert"?

Kinds of Premium Value

For illustration, let's imagine the free component of your offering is a basic car. We should choose the Everyman Car because one element of a profitable freemium is having a massive addressable market (preferably "every human on the planet" or "every business on the planet").

So what kinds of paid offerings can we provide to the drivers of our free vehicles? Let's look at 5, in order from most to least desirable and even dangerous.

Cargo Freemium

Your free car comes with 2 seats and a very small trunk. This is fine when you have no kids or 1 kid, but once you have 2 kids and their car seats and a stroller, trips to Costco become a circus of arranging and folding seats and jockeying diaper bags around your feet before heading to the warehouse for your 10 lbs. of oatmeal.

The paid offering says that for just $X per year you can add a roof rack or a trailer to your car (or a limousine extension), which will make all of these problems go away. Stubborn and frugal drivers will continue to struggle and shove and jam and limit themselves to the free capacity, but every week the pain comes again and every time you think of incrementing your offspring count you have to wonder if this is sane. Most people will eventually get the extra capacity.

Examples: Dropbox, iCloud.

Notice the change that takes place in the value of the free offering between 1 and 2 kids. At 0 and 1 child, the value of the free offering is constant. Once we reach 2, there's actually a decrease in the value of the free offering relative to the paid because of the constant hassle of dealing with the limitation. This is a subtle but critical way to rig the contest between the free and paid offering and help encourage people to convert.

Key characteristics:

- Inevitability of upgrade for most people as long as they keep using the product.
- The premium value is "more of the same" — the free car offering's primary benefit is moving people from point A to point B. The premium offering extends this primary benefit.
- Regular, repeated temptation to upgrade through normal usage of the product.
- Decrease in the value of the free offering (even) if user stays within the confines of free.

Ways this can go wrong:

- You choose the wrong dimension to limit and violate the "more of the same" requirement. It can be tempting to choose to throttle your user's storage space (bytes) but unless storage is the primary benefit / value of your product, you should choose a different dimension. (See comments at the end of this article about Flickr).
- You choose the right dimension but the wrong place to draw the line between free and paid.

Airbag Freemium

Every man wants a car, but every man has a different risk tolerance. You can sell him a car that has a certain level of safety (a minimum viable level, perhaps?), and offer him an upgrade that provides risk mitigation. Depending on what you offer, this premium value may offset risk only under extreme cases (e.g. a dedicated technician in the event of a server meltdown), or simply offer more around-the-clock safety and peace of mind (e.g. an uptime guarantee).

Example: Wordpress, Red Hat.

Key characteristics:

- Some people by nature will be interested in the premium offering. This might be through corporate or government mandate or in a B2C setting, a paranoia that requires it.
- Some people will be perfectly content with the free offering until something bad happens to them or someone they know.
- Some people will never upgrade because they like living on the edge.

Ways this can go wrong:

- Your product is used in a low-stakes setting where failure is OK. People don't upgrade.
- Your market doesn't contain enough people that will use the product in a high-stakes setting.
- Another company figures out how to sell the safety and security as an add-on that doesn't involve you.

Bomb Freemium

You know that scene in Speed where the bomb gets activated once the bus goes over 50 mph, and then they can't ever let it go below 50 mph or the bus will explode? Not everyone will want an airbag (see above), and not everyone will necessarily go 55 mph, but the people that do find themselves going 51+ mph will find themselves desperate to convert to your paid offering to defuse your well-designed bomb.

Example: Yammer, Salesforce to a lesser extent.

Most companies will never care that their employees are chatting away on yet another social media site, but some companies will see this as a disaster waiting to happen. Premium offering appeals to CIOs who can defuse the bomb by enacting controls that keep the usage of the product within certain acceptable/manageable boundaries.

Key characteristics:

- Product is easy to adopt in a high-stakes setting.

- Once the product has infected the high-stakes setting, its usage spreads virally.
- The more the product gets used, the more dependent the company becomes on its proper performance and usage.
- The premium value gives the company control over the product.

Ways this can go wrong:

- Your customers feel extorted.
- Your product never goes viral.
- Your product never gets used in a way that creates an exposed risk for the organization.

Cruise Control Freemium

A favorite feature of the long road trip is cruise control. Who wants to keep their foot pressed evenly on the gas for 3 hours at a time? Then again, if the driver only plans to use your car as a commuter, you'll never get him to buy in.

Example: GitHub.

Key characteristics:

- The free product is good enough for anyone that can avoid certain usage patterns or needs.
- Even the people that can't avoid these usage patterns or needs will not necessarily need the upgrade on day 1.

Ways this can go wrong:

- The people that can't avoid these usage patterns worm their way out of the pain or put up with it while it lasts.
- You don't retain users long enough for them to ever hit a point in time where they need cruise control.
- Your cruise control is a vitamin and not a painkiller.

Bells & Whistles Freemium

The fallback and default state for all freemium offerings (the state wherein you don't know enough about the users to know better) is to offer shiny things and upgrades that may be appealing but will never be a need for the vast majority of your users. In these cases your best chance is to turn the bells and whistles into needs (through marketing that convinces them of the great value) or to learn more about the market so you can graduate to a different kind of premium offering (see above).

Examples: pre-2012 Stormpulse (before we shifted to free trials/paid only) and many other startups.

Key characteristics:

- Startup founders don't know much about the market but understand that power users always exist.
- Premium offering tries to sell features to power users.
- Product gets clogged with features that "oooh" and "ahhh" but are very hard to quantify in terms of ROI.
- Marketing messages focus on the cost of running the business and the need to monetize to support operations rather than the value of the premium offering.

Risks:

- You run out of money because your conversion rate is low and your retention is poor (once people realize they don't need the upgrade they downgrade or don't renew).

Talking Points

What kind of freemium are you? Can you re-design your offering to get higher on the value ladder? Do you have enough funding to support yourself until your users start to convert? These are all important questions that will determine the fate of your business, and ignorance will not shield you from failure. ■

Matt Wensing is the Co-founder & CEO of Stormpulse, a web-based platform for companies to manage their weather risk. Matt grew up in South Florida, studied information design at the University of Chicago, and currently lives in Jupiter, Florida with his wife and four children.

Reprinted with permission of the original author.
First appeared in *hn.my/profree* (wensing.tumblr.com)

Why Cheap Customers Cost More

By SACHA GREIF

I was listening to Patrick McKenzie's podcast (with Amy Hoy as a guest) [hn.my/patio], and they touched on something that I had heard before: when you offer multiple plans for a service, the cheapest plan's customers tend to require the most support.

Now, at first this seems counter-intuitive. You'd expect the opposite: that the people who pay the most feel more entitled to support, and thus ask for more of it.

So if the cheapest customers truly require the most support, why is that? At first, I intuitively assumed it was a matter of character: like people who pinch hotel slippers and airline blankets, they simply wanted to extract the most value out of any situation. In other words, it's not so much that they need more support, they just abuse it because it's free.

The Better Explanation

But then I looked at my own example: when I do contact support, it's not because I enjoy it. It's because I don't have a choice. So I gave this matter more thought, and decided there's a better explanation.

I think cheap plans disproportionately attract a special category of users: dummies.

Now, being a dummy is not the same thing as being dumb. Being a dummy simply means that you're not well-versed in a particular domain, and you might benefit from reading a "* for dummies" book. In this way every one of us is a dummy at some things.

So let's assume I'm not well-versed in cars. I don't know the first thing about pistons, gearboxes, propellers, or drive shafts (can you tell I'm not pretending?). When it comes time to choose a car, what's the only rational factor that will drive (heh) my decision?

That's right, price.

And since I'm a dummy, when my car breaks down (or just runs out of gas…) I won't have a clue what happened and will be incapable of fixing the problem myself.

On the other hand, savvy customers consider many other factors besides price (otherwise, there wouldn't be much of a market for BMWs). And being savvy, these people are more likely to be able to troubleshoot their own problems.

So it's not that cheap people require more support. It's that people who require more support are more likely to make their purchasing decision based on price alone.

So once you've understood this, what can you do?

Educate Your Customers

Well, first of all you can try to educate your customers. Adding tooltips, wizards, help, and a FAQ will probably greatly reduce the amount of support requests you receive. After all, according to the 80/20 rule, the same 20% of issues probably cause 80% of requests.

Drive Dummies Away

Another possibility is to drive dummies away. In other words, make your service attractive to savvy customers who (like everybody else) are also looking for a bargain without attracting dummies in the process.

You can do that through "child-proofing": for example, Amazon's EC2 hosting offering is one of the cheapest around, but the amount of technical knowledge required to use it ensures that no dummy will inadvertently sign up.

Embrace Dummies

Lastly, you can embrace the dummies. This is a strategy used by Internet providers everywhere: advertise cheap prices to get that grandma demographic, but then make money on support and installation.

This strategy gets a bad rap because a lot of the companies who use it provide bad, expensive support. But there's no reason why you can't offer great expensive support.

Conclusion

No matter which strategy you choose, remember to consider the impact pricing will have on your customer base and your support costs. For example, a good strategy might be to start off with high prices, and then only lower them once you're ready to scale your support infrastructure.

In any case, remember that people simply respond to incentives, and that there's a very valid reason why your cheapest customers are asking the most questions! ■

Sacha is a user designer and entrepreneur from Paris, currently living in Kyoto. He has worked with startups such as Hipmunk, Codecademy, and Intercom. He's now focusing on Folyo, his own startup that helps companies find great freelance designers.

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

Everything I've Learned About Selling SaaS in Japan

By JASON WINDER

Japan is a notoriously difficult market to crack. Successful, established businesses entering Japan from overseas that do not bother to tailor their marketing and product for Japan regularly fail here.

Notably however, Japan's SaaS market is bigger than every other SaaS market in Asia combined. If you put in the time and effort to battle through the adversity, there is a wide range of fantastic opportunities here generated by criminally under-served market segments.

For the last 2 years, we've been building MakeLeaps, a tool to help businesses in Japan create, manage and send their invoices and quotes.

We soft-launched 7 months ago to almost zero fanfare — to our great relief. Unlike our other local competitors that have popped up in the last few months, we made an active decision to not issue press releases or seek media attention until we were confident we had both a deep understanding of our users requirements and a product they would find both useful and compelling.

Sending invoices and business documents in Japan is deeply ingrained with various cultural nuances and is ruled by a series of business manners and traditions. We had no idea if businesses would actually use an online system to send their invoices since you cannot get less "traditional" than using an online system to send invoices.

Having said that, 99% of all small/medium sized businesses in Japan are using Word and Excel to send their invoices. There's a strong argument that says "Word and Excel" are not exactly traditional Japanese software either, which gave us hope.

In the beginning, I would go to people's offices, put MakeLeaps in front of them on my laptop, and explain why we thought the software would be so great for them, both to practice my pitch and to garner some feedback. While many of these initial users were friends and acquaintances, I also cold-called a bunch of businesses, including accountants, to get deep level feedback about the system from as many actual potential users as possible.

After countless cycles of listening to these few initial customers I managed to get using the system and iterating on their feedback, things really started to take off around 2 months ago.

I feel like we're now in a good position to talk about our experiences and what we've learned.

You're a foreigner, and that's ok.

If you're a startup/company trying to enter Japan, you are very literally foreign in Japan. The kanji for foreigner literally translates to "outside." Being an "outsider" in the perspective of Japanese culture comes with drawbacks and benefits.

Benefits

- You have a degree of freedom because you're expected to be different. Japanese are understandably proud of their fantastic traditions and culture. They expect that most foreigners have no clue about the history, nuances and traditions that make up their country and, for the most part, they're correct.

This means that as a person attempting to introduce a product or service to Japan, you're not expected to be perfect in your knowledge of local customs, although of course, you're expected to try.

Drawbacks

- This "foreign-ness" is typically not a problem if you're selling a low-risk service, such as say, a social photo-taking application. In fact, it can even be cool for Japanese consumers to use a foreign application or tool. However, if you're selling a business-critical application, such as an invoicing tool, your foreign-ness becomes an issue.

  To a Japanese company, there are very real business and reputation risks in using a foreign tool for something as critical as invoicing. It's a human trait to trust people who are similar to you and who have shared experiences. It's important for you to discover whether or not your foreign-ness is an issue.

Here's how we got around this:

- One strategy we're using, is the "feature what you can't fix" idea. On our Q&A page we make it very clear, who we are (essentially, not Japanese!), why we're in Japan, and why we're working in this problem space.

Since this is a big issue, I go into further detail on various additional strategies we've used to beat this in the section below titled "Japanese consumers make buying decisions in different ways."

It's really, really hard to get feedback.

The concept of social harmony and saving face in Japan is baked into every relationship in Japan, whether they're personal or business. In fact in Japan, there's often no difference between these two concepts.

This creates a tricky dichotomy where:

1. A user with no personal connection to you will feel no obligation to providing you with feedback, preferring to simply close the tab and move on if software doesn't suit their needs.
2. A user with a personal connection to you will rarely provide you with "real" feedback because that could disturb the social harmony in your relationship and could cause you to lose face. Both terrible outcomes for your thoughtful Japanese friend.

This means that it's almost impossible to garner feedback in Japan.

Here's how we got around this:

- We openly ask for comments in our welcome email, our support page, and also on Twitter and Facebook.
- We often email regular users and new users from a "real person," asking how things are going and politely requesting any feedback or comments the user might have.
- We do our best to build relationships with our users who are bilingual. The idea of dissenting opinions and openly discussing/disagreeing with people is typically more accepted overseas and in international companies. We find that bilingual users with experience in international companies are often quite open to sharing their feedback and comments.
- When a user does provide us with feedback, we:
  - Instantly reply to them explaining how grateful we are to receive this feedback.
  - Explain that we take their feedback and comments very seriously.
  - We explain that even though we cannot implement every suggestion, we promise to discuss it with the team and get back to them if there's an update on their issue.

This approach hopefully communicates our willingness to listen to feedback and leaves the door open for future communication.

Japanese consumers make buying decisions in different ways.

Japan has a culture based on personal networks and connections, so Japanese consumers typically look for consensus and poll their network while making a buying decision. This can lead to a pleasant snowball effect, where if enough customers start using a product/service, critical mass kicks in and the product/service achieves extraordinary success.

A recent example is Apple. You can't get on a train and sit down without seeing 20-30+ iPhones, iPads or white earbuds leading to iPhones/iPads/iPods.

The flip-side of this is that no matter how good your product/service is, people won't use it unless they see other people using it, or unless their friends introduce them to it.

If it's a business service, this is doubly true since there's real risk in using a "non-consensus approved" product/service.

How we got around it:

This is a big one, so we've come up with a lot of strategies to battle this.

- Avoid the English/Japanese content problems by creating separate social media accounts that are 100% Japanese for your potential customers in Japan.
- We don't recommend you begin in Japan with posting blog content. We had one very popular article on the MakeLeaps blog that was shared a lot in Japan, and we got about 60,000 views. Grand result: 0 signups. We haven't given up on blog content and we've got a bunch of ideas on how to increase conversion here, but I recommend building up to blog content instead of starting with it.
- When a user talks about us publicly, we feature it on our Facebook page with a public and personal message from us to them, saying, "Thank you very much for using the service. We're really happy you're getting benefit from it!" It has been critically important to show Japanese businesses that other Japanese businesses are using our software.
- To further reinforce the idea that many companies in Japan are using our service, we printed MakeLeaps stickers and made them available for free to all our users. When a user is nice enough to send a photo of their PC/Mac/iPad with the MakeLeaps sticker, we upload the photo to Facebook.
- We've had a lot of success with our Facebook page in Japan. Facebook is exploding in Japan, and to some degree we've managed to ride this wave of growth. Facebook is our primary tool for regular communication with our users.
- We do everything we can to promote our regular users, which benefits us both. We offer many of our regular users the following mini-package:
  - Featured quote on the front page.
  - Featured blog post where we come to their office, ask them some questions, take some photos, publish it on our blog and promote it by FB/Twitter/Blog/Newsletter.
- We treat every customer support ticket with the utmost importance. Not only is it good business practice, but the business benefit of one "happy tweet"(Japanese) from a satisfied customer far outweighs the effort on our side to ensure they're extraordinarily happy with the support they receive.

Japanese consumers are extremely sensitive to "Japanese-ness." And rightfully so.

Ever read something "Google Translated" into your language? In Japanese, it sounds about 10 times worse.

To a Japanese consumer, if you can't get basics like the language copy right, chances are pretty good you're not going to get the more fundamental things right either, like the value proposition to Japanese consumers.

How we get around this:

- There is no way around this. You need staff on the ground in Japan who can communicate effectively to your customers through the website copy, and the language in the application itself. Not to mention collecting feedback on the features and the application itself.
- Another tip: Japanese consumers favor websites with very dense content, and little whitespace. This hurts the latte drinking minimalist designer inside us, but we're not here to argue.

Here are some examples of very popular Japanese sites:

Here's the more "spaced out" English MakeLeaps Site:

Cloud Invoicing and Quoting Software

Here's the more dense Japanese MakeLeaps site:

"Insanely great" customer service.

First-time visitors to Japan are always amazed at the incredible service they receive here. The service levels are probably the best in the world.

The thing is that your customers will expect a similar level of service from you. Failure to provide this high level of service gives your customer an instant feeling of disconnection from you, and you're quickly cemented into the "outsider" box since you don't get the Japanese customs regarding customer service.

How we get around this:

- We do our best to provide a fantastic level customer service (in both English and Japanese).
- We provide a phone number for people to call. Interestingly, we are currently the only online invoicing tool in Japan that does this. Even more interestingly, we get a lot of phone calls to this number.
- All of our sales/support staff are on Twitter. We tried connecting with our customers through the @MakeLeaps_JP twitter account, with very little success. Once we changed to personal twitter accounts, our customer engagement on Twitter shot up.
- Another avenue where we've had a lot of success is having direct founder engagement with users. I've had some very pleasant conversations from surprised users when I've sent tweets in Japanese from @JasonWinder.

Campaign!

Japanese companies commonly use the concept of campaigns when marketing. We've had a lot of success with the following 2 campaigns:

- **Our "Spring" campaign**: Offering incentives for: providing us feedback we can use to improve our system, introducing their friends to MakeLeaps, and tweeting about us.
- **A 24 Hour signup campaign**: We provide our Facebook followers with a registration code that provides them with 3 free "mail points." A mail point allows users to have one invoice printed and sent by Japan Post by us. We saw a huge spike in signups during and after this 24-hour campaign.

Summary

Much of what we've learned has only been possible because we're on the ground and able to directly contact and interact with our customers.

For a SaaS product targeting the Japanese market, you will absolutely need a team on the ground to get you the information you need for your build/measure/learn cycle because success in Japan is very rarely achieved accidentally.

In our experience, your realistic options are either to build a team in Japan or to partner with a company in Japan experienced in market entry. Although typically, companies that offer market entry services have price tags that are prohibitive for early stage startups. ■

Jason Winder has been in Japan since 2001, and set up an IT services business in Tokyo in 2003 called Webnet IT. After getting snowed in by general admin work and sending an ever increasing number of quotes and invoices every month, he built some software to automate some of these workflows such as invoicing, payroll, expenses and reporting. In 2010, Jason co-founded a SaaS startup catering to businesses in Japan called MakeLeaps.

Reprinted with permission of the original author.
First appeared in *hn.my/japan* (makeleaps.jp)

Consistent Hashing

BY TOM KLEINPETER

Let's say you're a hot startup and your database is starting to slow down. You decide to cache some results so that you can render web pages more quickly. If you want your cache to use multiple servers (scale horizontally, in the biz), you'll need some way of picking the right server for a particular key. If you only have 5 to 10 minutes allocated for this problem on your development schedule, you'll end up using what is known as the naïve solution: put your N server IPs in an array and pick one using key % N.

I kid, I kid — I know you don't have a development schedule. That's OK. You're a startup.

Anyway, this ultra simple solution has some nice characteristics and may be the right thing to do. But your first major problem with it is that as soon as you add a server and change N, most of your cache will become invalid. Your databases will wail and gnash their teeth as practically everything has to be pulled out of the DB and stuck back into the cache. If you've got a popular site, what this really means is that someone is going to have to wait until 3am to add servers because that is the only time you can handle having a busted cache. Poor Asia and Europe — always getting screwed by late night server administration.

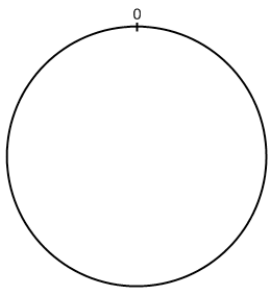
You'll have a second problem if your cache is read-through or you have some sort of processing occurring alongside your cached data. What happens if one of your cache servers fails? Do you just fail the requests that should have used that server? Do you dynamically change N? In either case, I recommend you save the angriest tweets about your site being down. One day you'll look back and laugh. One day.

As I said, though, that might be OK. You may be trying to crank this whole project out over the weekend and simply not have time for a better solution. That is how I wrote the caching layer for Audiogalaxy searches, and that turned out OK. The caching part, at least. But if had known about it at the time, I would have started with a simple version of consistent hashing. It isn't that much more complicated to implement, and it gives you a lot of flexibility down the road.

The technical aspects of consistent hashing have been well explained in other places, and you're crazy and negligent if you use this as your only reference. But, I'll try to do my best. Consistent hashing is a technique that lets you smoothly handle these problems:

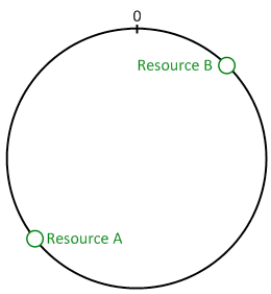
- Given a resource key and a list of servers, how do you find a primary, second, tertiary (and on down the line) server for the resource?
- If you have different size servers, how do you assign each of them an amount of work that corresponds to their capacity?
- How do you smoothly add capacity to the system without downtime? Specifically, this means solving two problems:
  - How do you avoid dumping 1/N of the total load on a new server as soon as you turn it on?
  - How do you avoid rehashing more existing keys than necessary?

In a nutshell, here is how it works. Imagine a 64-bit space. For bonus points, visualize it as a ring, or a clock face. Sure, this will make it more complicated when you try to explain it to your boss, but bear with me:

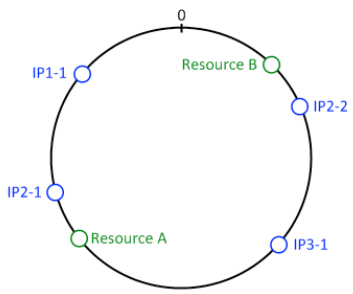


That part isn't very complicated.

Now imagine hashing resources into points on the circle. They could be URLs, GUIDs, integer IDs, or any arbitrary sequence of bytes. Just run them through a good hash function (e.g., SHA1) and shave off everything but 8 bytes. Now, take those freshly minted 64-bit numbers and stick them onto the circle:



Finally, imagine your servers. Imagine that you take your first server and create a string by appending the number 1 to its IP. Let's call that string IP1-1. Next, imagine you have a second server that has twice as much memory as server #1. Start with server #2's IP, and create 2 strings from it by appending 1 for the first one and 2 for the second one. Call those strings IP2-1 and IP2-2. Finally, imagine you have a third server that is exactly the same as your first server, and create the string IP3-1. Now, take all those strings, hash them into 64-bit numbers, and stick them on the circle with your resources:



Can you see where this is headed? You have just solved the problem of which server to use for resource A. You start where resource A is and head clockwise on the ring until you hit a server. If that server is down, you go to the next one, and so on and so forth. In practice, you'll want to use more than 1 or 2 points for each server, but I'll leave those details as an exercise for you, dear reader.

Now, allow me to use bullet points to explain how cool this is:

- Assuming you've used a lot more than 1 point per server, when one server goes down, every other server will get a share of the new load. In the case above, imagine what happens when server #2 goes down. Resource A shifts to server #1, and resource B shifts to server #3 (Note that this won't help if all of your servers are already at 100% capacity. Call your VC and ask for more funding).
- You can tune the amount of load you send to each server based on that server's capacity. Imagine this spatially: more points for a server means it covers more of the ring and is more likely to get more resources assigned to it.

  You could have a process try to tune this load dynamically, but be aware that you'll be stepping close to problems that control theory was built to solve. Control theory is more complicated than consistent hashing.
- If you store your server list in a database (2 columns: IP address and number of points), you can bring servers online slowly by gradually increasing the number of points they use. This is particularly important for services that are disk bound and need time for the kernel to fill up its caches. This is one way to deal with the datacenter variant of the Thundering Herd Problem [hn.my/thunder].

  Here I go again with the control theory — you could do this automatically. But adding capacity usually happens so rarely that just having somebody sitting there watching top and running SQL updates is probably fine. Of course, EC2 changes everything, so maybe you'll be hitting the books after all.
- If you are really clever, when everything is running smoothly you can go ahead and pay the cost of storing items on both their primary and secondary cache servers. That way, when one server goes down, you've probably got a backup cache ready to go.

  Pretty cool, eh?

I want to hammer on point #4 for a minute. If you are building a big system, you really need to consider what happens when machines fail. If the answer is "we crush the databases," congratulations: you will get to observe a cascading failure. I love this stuff, so hearing about cascading failures makes me smile. But it won't have the same effect on your users.

Finally, you may not know this, but you use consistent hashing every time you put something in your cart at Amazon.com. Their massively scalable data store, Dynamo [hn.my/dynamo], uses this technique. Or if you use Last.fm, you've used a great combination: consistent hashing + memcached. They were kind enough to release their changes, so if you are using memcached, you can just use their code without dealing with these messy details. But keep in mind that there are more applications to this idea than just simple caching. Consistent hashing is a powerful idea for anyone building services that have to scale across a group of computers. ■

Tom Kleinpeter is the CTO and cofounder of Audiogalaxy, where he is working to build a better experience for music lovers. He lives near Seattle with his family.

Relational Shell Programming

By MATT MIGHT

No one would mistake the average shell script for principled software.

Yet, if we look at how scripts are used, patterns emerge.

Unix is a bestiary of *ad hoc* databases: comma-, colon-, tab- and space-separated tables. Think of `/etc/*` or `/var/log/*`, or of columnar commands.

Shell scripts commonly, if unknowingly, compose five (of six) primitive relational-algebraic operations on these tables: union, difference, projection, selection, and renaming:

- `cat` acts like union
- `sed` and `grep` act like selection
- `cut` acts like projection
- `awk` can perform renaming
- `diff` acts (almost) like difference

Relational algebra (whose sixth primitive operation is Cartesian product) is equivalent to both relational calculus and SQL.

Cartesian product (and equijoin) are not difficult to create in bash.

If you find yourself stumbling into a relational design pattern in a shell script, consider making that relationality rigid and explicit.

Read on to learn a little more about databases, shell scripts, or both.

Representing relations

In mathematics, a relation is a set of tuples.

[A tuple is an ordered collection of values, (v1,…,vn).]

For example, {(Bob,31),(Judy,32)} is a relation.

In database theory, a relation is a set of tuples with an assigned name for each column; that is, a relation is a table.

For the earlier relation, we could define a header tuple (name,age) that names each column.

We could then represent the relation explicitly as a table:

name	age
Bob	31
Judy	32

Relational algebra is a theory for manipulating relations whose power is equivalent to SQL and relational calculus.

Remarkably, relational algebra has only six primitive operations.

I define the six primitives below, but if you're looking for a comprehensive work on relational theory, particularly as it relates to modern databases, I recommend Date's SQL and Relational Theory [hn.my/sqlrel].

Relations in Unix

Many Unix commands interpret files like relations: each line is a tuple.

CSV or "comma-separated value" files are a crude, generic way to encode a relation. For example, see the MaxMind GeoIP database [hn.my/maxmind]:

```
"1.0.0.0","1.0.0.255","16777216","16777471","Australia"
"1.0.1.0","1.0.3.255","16777472","16778239","China"
```

The password and group files for a Unix system (`/etc/passwd` and `/etc/group`), which store account and login information, are relations with elements of individual tuples separated by colons (`:`):

```
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
```

The /etc/hosts file, which can fix a hostname to an IP address, is a space-separated relation:

```
127.0.0.1        localhost
255.255.255.255  broadcasthost
::1              localhost
fe80::1%lo0      localhost
```

The /etc/resolv.conf file, which specifies domain name information, is a space-separated relation:

```
domain hsd1.ut.comcast.net.
nameserver 208.67.220.220
nameserver 208.67.222.222
```

The command netstat -nr produces tables of routing information.

```
Destination    Gateway    Flags    Refs      Use  Netif
127            127.0.0.1  UCS         0        0    lo0
127.0.0.1      127.0.0.1  UH          5   151051    lo0
224.0.0/4      lo0        UmCS        1        0    lo0
224.0.0.251    lo0        UHmW3I      0        0    lo0
```

The command traceroute produces relation-like data:

```
5  69.139.247.14  16.560 ms  26.933 ms  11.521 ms
6  68.86.90.125   74.433 ms  40.330 ms  57.565 ms
7  68.86.87.30    54.590 ms  52.155 ms  55.322 ms
```

Many log files are, in effect, giant relations as well.

Union

Mathematically, union (∪) combines the contents of two sets:

$$A \cup B = \{ x \mid x \in A \quad or \quad x \in B \}$$

In the shell, union is straightforward — it's `cat`:

```
$ cat relation-1 relation-2 >
unioned-relation
```

After any operation in which duplicate entries are a possibility, compose with `sort` and `uniq`:

```
$ cat relation-1 relation-2 | sort |
uniq > unioned-relation
```

Since a relation is a set, duplicate entries are not allowed.

Many other programs accept multiple files as arguments and print out their contents: `sed`, `grep`, `cut`, `awk`.

Thus, these commands can combine selection (and projection) with union.

To "union" the output of one command with another, use a sequencing semicolon with an append:

```
$ command-1 > relation; command-2 >>
relation
```

Selection

Mathematically, selection (σΦ) is a filtering operation that applies a predicate Φ to each member of a set, retaining only those for which the predicate is true:

$$\sigma \Phi (A) = \{ x \mid x \in A \quad and \quad \Phi (x) \quad is\ true \}$$

Many Unix commands serve as selectors (and simultaneously as union); that is, many programs can filter a file, line by line.

The most common selectors are `sed`, `grep`, and `awk`.

For example, `awk` can select entries from `/etc/passwd` where the user ID and the primary group ID are not equal:

```
awk -F ":" '{ if ( $3 != $4 ) print }'
/etc/passwd
```

Projection

Mathematically, projection ($\pi_{c_1,\ldots,c_n}$) is a reduction operator that retains the specified columns ($c_1,\ldots,c_n$) in a relation and discards the rest.

If a relation is table, then $c_1,\ldots,c_n$ can be the names of the columns (and the order in which to preserve them).

Otherwise, $c_1,\ldots,c_n$ can be column numbers so that:

$$\pi_{i_1,\ldots,i_n}(R)=\{(x_{i_1},\ldots,x_{i_n})\mid(x_1,\ldots,x_n)\in R\}$$

The Unix command cut is a projection-like command.

For example, cut can project just the user names and shells out of `/etc/passwd`:

```
$ cut -d ":" -f 1,7 /etc/passwd
```

The Unix commands sed and awk also behave like projectors.

For example, cut cannot reorder columns, but awk can:

```
$ awk -F":" '{ print $7 ":" $1 }' /etc/passwd
```

If the syntactic structure of each tuple is more complex, then sed can pull it apart with a regular expression.

Rename

For tables, the rename operation ($\rho_{c/c'}$) changes the name of column c' to c.

Since most Unix tables don't specify headers, or the structure is implicit, there is no equivalent of or need to rename.

Columns are implicitly named positionally.

`awk` can easily reorder columns.

Cartesian product

Cartesian product is the source of much power for relational algebra, since it is the key ingredient in joins — the primitive operation that combines two tables together.

Relational Cartesian product (χ) fully combines two relations:

$$A\chi B=\{(a_1,\ldots,a_n,b_1,\ldots,b_m)\mid(a_1,\ldots,a_n)\in A \text{ and } (b_1,\ldots,b_m)\in B\}$$

Surprisingly, there is no shell equivalent of Cartesian product.

Fortunately, it's not hard to write a `cartesian` script:

```
#!/bin/bash

delim="," # CSV by default

# Parse flagged arguments:
while getopts "td:" flag
do
  case $flag in
    d) delim=$OPTARG;;
    t) delim="\t";;
    ?) exit;;
  esac
done

# Delete the flagged arguments:
shift $(($OPTIND -1))

# Remaining args now in $*

# Now join $1 and $2
while read a;
 do while read b;
      do echo "$a${delim}$b";
      done < $2;
 done < $1
```

Difference

Mathematically, set difference (–) removes the elements of a set that are present in another set:

$$A-B=\{a\mid a\in A \text{ and } a\notin B\}$$

The command `diff` reports the difference between two files, but it does not perform relational set difference.

We need to implement relational set difference.

It is simpler to implement `difference` if we assume the existence of a membership tester, `memberp`.

The command `memberp file line` exits successfully if the file *file* contains the line *line*, and it fails otherwise.

With access to `memberp`, `difference` becomes:

```
#!/bin/bash

while read a;
 do
   if ! memberp $2 "$a"; then
     echo "$a"
   fi
 done < $1
```

Membership

A membership script, `memberp`, searches line by line:

```
#!/bin/bash

# Usage: memberp file line

file=$1

shift

while read a;
 do
   if [ "$a" = "$*" ]; then
     exit 0
   fi
 done < $file

exit -1
```

Join

A join is a synthetic operation composed of selection, projection, and product.

A common join is `equijoin`, where the product of two relations is filtered based on the equality of two columns.

To perform an `equijoin`, we need two relations and columns to compare.

The command `equijoin [ -d <delim> | -t ] rel1 rel2 col1 col2` uses `awk` to combine and compare two tables:

```
#!/bin/bash

delim="," # CSV by default

# Parse flagged arguments:
while getopts "td:" flag
do
  case $flag in
    d) delim=$OPTARG;;
    t) delim="\t";;
    ?) exit;;
  esac
done

# Delete the flagged arguments:
shift $(($OPTIND -1))

f1=$1
f2=$2
col1=$3
col2=$4

cols=`awk -F "${delim}" '{ print NF ; exit }'
$f1`

while read a
 do while read b
      do
        echo "$a${delim}$b"
      done < $f2
 done < $f1 |
 awk -F "${delim}" "{ if ( \$${col1} == \$$((
col2 + cols )) ) print }"
```

Example: Deleting a list of bad users

We're given a list of bad users to remove from `/etc/passwd`.

They're in the file `bad.db`:

```
matt
bob
```

Let's use a simplified `/etc/passwd` for this example:

```
root:*:0:0:The Admin:/root:/bin/sh
matt:*:500:500:Matt:/home/matt:/bin/bash
john:*:501:501:John:/home/john:/bin/bash
bob:*:502:502:Bob:/home/bob:/bin/bash
```

How can we delete them relationally?

Start with a product, `cartesian -d ":" bad.db /etc/passwd > badpasswd.db`:

```
matt:root:*:0:0:The Admin:/root:/bin/sh
matt:matt:*:500:500:Matt:/home/matt:/bin/bash
matt:john:*:501:501:John:/home/john:/bin/bash
matt:bob:*:502:502:Bob:/home/bob:/bin/bash
bob:root:*:0:0:The Admin:/root:/bin/sh
bob:matt:*:500:500:Matt:/home/matt:/bin/bash
bob:john:*:501:501:John:/home/john:/bin/bash
bob:bob:*:502:502:Bob:/home/bob:/bin/bash
```

Then, select rows where the bad user matches the account name, with `awk -F: '{ if ( $1 == $2 ) print }' badpasswd.db > offenders.db` :

```
matt:matt:*:500:500:Matt:/home/matt:/bin/bash
bob:bob:*:502:502:Bob:/home/bob:/bin/bash
```

Now, project out the columns that make the result compatible with the format of `/etc/passwd` with `cut -d ":" -f2- offenders.db > kill.db`:

```
matt:*:500:500:Matt:/home/matt:/bin/bash
bob:*:502:502:Bob:/home/bob:/bin/bash
```

Finally, use difference /etc/passwd kill.db to generate a new password file sans malefactors:

```
root:*:0:0:The Admin:/root:/bin/sh
john:*:501:501:John:/home/john:/bin/bash
```

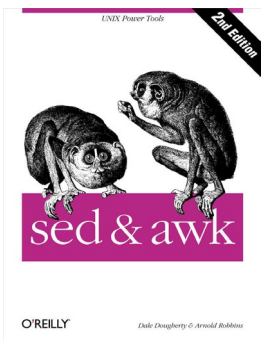
Exercise

Some of these scripts have quadratic time complexity.

- Add a `-s` flag to `equijoin` and `difference` that assumes the inputs have been sorted by `sort` and produces a sorted output.
- Show that `-s` can achieve better time complexity.
- Then, rewrite the account-deletion example to eliminate Cartesian product and use fast `equijoin` instead.

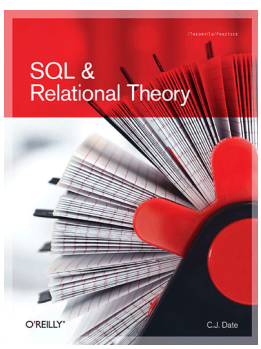
Good resources

Sed and awk have become a lost art. They're the only languages I know that frequently beat Perl in semantic density. If you haven't learned them yet, you'll be impressed with what they can do for you.



I recommend the O'Reilly classic, sed & awk [hn.my/sedawk].

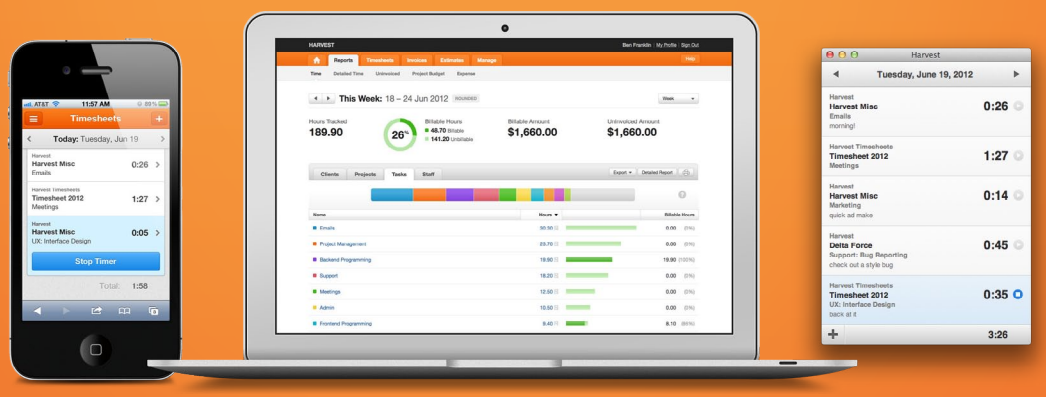
I also recommend spending an hour or two going through the advanced bash scripting guide [hn.my/abs] and bash pitfalls [hn.my/pitfalls]. It's worth the time invested.



If you want a solid theoretical understanding of databases, I recommend Date's SQL and Relational Theory [hn.my/sqlrel]. ■

Matt Might is a professor of Computer Science at the University of Utah. His research interests include programming language design, staticanalysis and compiler optimization. He blogs at *matt.might.net/articles* and tweets from @mattmight

Reprinted with permission of the original author. First appeared in *hn.my/sqlshell* (matt.might.net)



HARVEST

Still using that rusty old Perl time tracking script you wrote when Reagan was still in office? Try Harvest for two weeks and let us show you a better way to track time and get paid.

getHarvest.com/hackers

Williams, Master of the "Come From"

By REGINALD BRAITHWAITE

In this business, you meet more than your fair share of eccentric developers with their own idiosyncratic ideas about software development. If you're like me, you need go no further than a mirror to see what one looks like. One of the most iconoclastic fellows I ever met was a grizzled veteran named Jim Kelly. For reasons of interest only in cultural antiques, everyone called him "Williams."

Williams claimed to have gotten started when using punch cards and random access memory was considered effete by the drum memory wizards of the previous generation [hn.my/mel]. Like many retro-grouches, he would fondly recall rites of passage, such as booting computers by toggling the three bootstrap instructions into a CPU's front panel. If his eccentricity stopped at recounting tales from days of yore, I'd probably have forgotten him years ago.

However, Williams was unique. Most veterans cling to whatever beliefs about software development were in vogue when they were cutting their teeth. Be it the utility of Lisp, the superiority of the VAX architecture, or the string-theory-like perfection of Smalltalk, most veterans cling to technology and ideas that were cutting edge back in their day. Not Williams. Williams had invented his own software development methodology, and he espoused it with the fervor of an evangelist on a street corner.

He had long ago accepted that the rest of the world was not about to change to do things the "right way," but he likewise refused to submit to fashion, no matter what the costs to his reputation or career. So he bounced from job to job, constantly being let go for "lack of fit with the team," until he wound up at ThinkWare, a contracting firm. The partners at ThinkWare specialized in finding ways for square pegs to write software that fit in round holes, and they carefully walled Williams off so that he could write software his own way with very little interference from anyone else.

Williams had a style that ThinkWare described as "unorthodox, but effective."

I'll take you, darling. And you. And you. And. You.

Typically, ThinkWare would give Williams a list of features to implement. Williams was very big on dependencies, and he would very carefully draw a graph of feature dependencies. For example, if he was going to write a Facebook Clone, his diagram would show that commenting on a wall post depended on wall posts, which in turn depended on walls.

Williams would then work along the graph, not starting any feature until all of its dependencies were fully implemented. And he would work on a single feature at a time, completing the feature in its entirety before starting another. Williams loathed developing infrastructure that would "pay off later;" he preferred to write the simplest and least amount of code required to get the current feature to work.

In that respect, his style resembled that of many agile developers. And in another respect, he appeared "agile" to those who never read his code: Williams loved to write tests. When Williams delivered a feature, there was much, much more test code than working code. He had been taught to write automated unit tests at a time when the standard architecture for making code "testable" consisted of

writing a command-line wrapper so that the tests could be baked into a shell script, and he never wavered from his belief that if you haven't tested it, it doesn't work.

So Williams' process was simple: Work on features one at a time, write the minimum amount of code to get the feature to work, practice YAGNI, and write comprehensive tests for each feature.

Too busy looking good.

Although Williams' style seemed contemporary, it was his heterodox practices that drove a wedge between him and his colleagues. Unlike every other agilist on this or any planet, Williams disdained refactoring. It wasn't that he saw no value in refactoring: Scrolling through his check-ins revealed that he would write and rewrite the code for each feature numerous times.

However, once Williams delivered a feature, he liked to move on to the next feature and leave the code for existing features unchanged as much as possible. For example, if he delivered a wall post feature, it might include an ActiveRecord model class:

```
# wall_post.rb
class WallPost < ActiveRecord::Model
    # ...
end
```

Once the Wall Posts were delivered with tests, he might work on comments. But instead of changing `wall_post.rb` to read:

```
# wall_post.rb
class WallPost < ActiveRecord::Model
    has_many :comments
    # ...
end
```

Williams would isolate all of the code for comments into a module or plug-in, and "monkey patch" the WallPost class:

```
# vendor/plugins/comments/lib/railtie.rb
WallPost.class_eval do
    has_many :comments
    # ...
end
```

Since none of the code written so far needed to know that a wall post could have comments, he saw no value in cluttering up those files with comment-handling code. Instead, he put the relationship between wall posts and comments in the code that was responsible for doing something with comments. His code was uniformly organized so that the code dependencies were exactly isomorphic to the feature dependencies.

Williams used every decoupling technique in the book and several he invented himself, from monkey-patching to method combinators to writing observers on classes. Inexperienced developers would often be completely bewildered as to how anything worked at all and would search in vain for signs of a heavyweight Dependency-Injection framework.

Features are fairly coarse-grained, so after getting over the shock of Williams' style, most developers could adjust. However, Williams also used these decoupling techniques for fine-grained cross-cutting concerns as well. So instead of writing:

```
# wall_post.rb
class WallPost < ActiveRecord::Model
    def doSomething
        WallPost.transaction do
            # ...
        end
    end
end
```

Williams would write:

```
# wall_post.rb
class WallPost < ActiveRecord::Model
    def do_something
        # ...
    end
end
```

And:

```
# wall_post_persistence.rb
WallPost.class_eval do
    def do_something_with_db_transaction
        WallPost.transaction do
            do_something_without_db_transaction
        end
    end
    alias_method_chain :do_something, :db_trans-
action
end
```

The net result was that his models were always small and directly concerned with business logic, while implementation details like error handling and persistence were moved out into separate modules and plug-ins.

How sloppy your man works.

Of course, his colleagues rioted at the thought of working with his code.

> *In Smalltalk, everything happens somewhere else. –Adele Goldberg*

When looking at one of Williams' model classes, all you would see is the basic, bare bones minimum functionality. Other features would be entirely implemented elsewhere, often in plug-ins of one kind or another. That made following the code annoying for developers used to the "one model class to rule them all" style of coding.

And the one or two folks who prayed at the Big Church [springsource.org] were aghast at the lack of an Injection Container and the complete absence of XML configuration. As one educated wag put it:

> *Williams has elevated the computed, non-local COMEFROM to an art form.*

Worse, Williams' style made it obvious how little work he really did. His methods were short and simple and to the point. It looked as if he was bored and making work for himself with all the decoupling because the decoupling was easily the most advanced code in any of his projects.

It seemed like a code smell to have infrastructure code that was more advanced than the domain code, as if he was an architecture astronaut intent on making life hard for himself and his colleagues.

Unorthodox, but effective.

Meanwhile, the ThinkWare partners had adopted a tolerant attitude towards Williams and his code. Despite the objections from his colleagues, they had noticed that his code really did work and kept working, mostly because it really was decoupled. New features could always be backed out simply by removing their plug-ins or modules.

This was most evident when examining his mountains of test code. Very little of it was concerned with mocking and stubbing functionality because he could always test his models without persistence layers or error handling frameworks to stub in.

Williams, they had discovered, was a good developer...within his limitations. Despite how well he worked and how well his code worked, his dependency decoupling would invariably erode when handed off to someone else. Entropy would rot his architecture as developers would work around it to get things done quickly.

Ghettoes are the same all over the world.

It wasn't that his colleagues were unimaginative or lazy, but Williams' style clashed with an environment that celebrates YAGNI and refactoring. Decoupling was, most developers reasoned, something they weren't going to need, and if they did, it could be re-factored in later.

So the ThinkWare partners put Williams in charge of small projects where he could practice his own brand of architecture in peace and largely left him to his own devices. The clients were happy, and he was happy. Once in a while another developer would work with him on a limited basis, and after getting up to speed on how things worked in his particular corner of the universe, they'd fall in line and get a lot of stuff done.

Sometimes they'd return to another project and try to implement some of his ideas, with middling success. But as long as the developers learned something from the experience, the ThinkWare partners figured that pairing Williams with a colleague from time to time was a win.

The last time I saw Williams, he had grown an afro and was carrying a tennis racket, obviously on his way to a game. We chatted for a while, and he excitedly told me about a framework he was developing for implementing really lightweight decoupling in some weird dialect of JavaScript.

I never saw him again, but I like to imagine that he's still at ThinkWare, writing solid code and evangelizing his ideas to anyone who will listen. ■

Reginald is a software developer and development lead with Unspace Interactive. He writes code and words about code in homoiconic. Follow him on Twitter @raganwald

Reprinted with permission of the original author. First appeared in *hn.my/comefrom*

Why is the DOS Path Character "\"?

By LARRY OSTERMAN

MANY, MANY MONTHS ago, Declan Eardly asked why the \ character was chosen as the path separator.

The answer is from before my time, but I do remember the original reasons.

It all stems from Microsoft's relationship with IBM. For DOS 1.0, DOS only supported floppy disks.

Many of the DOS utilities (except for command.com) were written by IBM, and they used the "/" character as the "switch" character for their utilities. The "switch" character is the character that's used to distinguish command line switches: on *nix, it's the "-" character, on most DEC operating systems (including VMS, the DECSystem-20 and DECSystem-10), it's the "/" character.

The fact that the "/" character conflicted with the path character of another relatively popular operating system wasn't particularly relevant to the original developers — after all, DOS didn't support directories, just files in a single root directory.

Then along came DOS 2.0. DOS 2.0 was tied to the PC/XT, whose major feature was a 10M hard disk. IBM asked Microsoft to add support for hard disks, and the MS-DOS developers took this as an opportunity to add support for modern file APIs. They added a whole series of handle-based APIs to the system (DOS 1.0 relied on an application-controlled structure called an FCB). They also had to add support for hierarchical paths.

Now historically there have been a number of different mechanisms for providing hierarchical paths. The DecSystem-20, for example represented directories as: "<volume>:"<"<Directory>[.<Subdirectory>">"FileName.Extension[,Version]" ("PS:<SYSTEM>MONITR.EXE,4"). VMS used a similar naming scheme, but instead of < and > characters it used [and] (and VMS used ";" to differentiate between versions of files). *nix defines hierarchical paths with a simple hierarchy rooted at "/". In *nix's naming hierarchy, there's no way of differentiating between files and directories, etc (this isn't bad, btw, it just is).

For MS-DOS 2.0, the designers of DOS chose a hybrid version. They already had support for drive letters from DOS 1.0, so they needed to continue using that. And they chose to use the *nix style method of specifying a hierarchy: instead of calling the directory out in the filename (like VMS and the DEC-20), they simply made the directory and filename indistinguishable parts of the path.

But there was a problem. They couldn't use the *nix form of path separator of "/", because the "/" was being used for the switch character.

So what were they to do? They could have used the "." character like the DEC machines, but the "." character was being used to differentiate between file and extension. So they chose the next best thing, the "\" character, which was visually similar to the "/" character.

And that's how the "\" character was chosen.

Here's a little known secret about MS-DOS. The DOS developers weren't particularly happy about this state of affairs. Heck, they all used Xenix machines for email and stuff, so they were familiar with the *nix command semantics. So they coded the OS to accept either "/" or "\" character as the path character (this continues today, by the way . Try typing "notepad c:/boot.ini" on an XP machine if you're an admin). And they went one step further. They added an undocumented system call to change the switch character. And updated the utilities to respect this flag.

And then they went and finished out the scenario: they added a config.sys option, SWITCHAR= that would let you set the switch character to "-".

Which flipped MS-DOS into a *nix style system where command lines used "-switch", and paths were / delimited.

I don't know the fate of the switchar API, it's been long gone for many years now.

So that's why the path character is "\". It's because "/" was taken. ■

Larry Osterman is a Principal Software Design Engineer at Microsoft where he has worked since 1984. He lives in the Seattle area.

Reprinted with permission of the original author.
First appeared in *hn.my/dos* (blogs.msdn.com)

Being Deaf

By DAVID PETER

At 21, I'm the youngest employee at the startup 1000memories. I'm also their first deaf employee. At a startup, I likely always will be the first. For a startup to succeed, the team must communicate well together. Since I can't hear, that presents a major challenge for my employers.

On top of that, there are certain things about being deaf that people have never considered, understate, or are mistaken about — so I must clear up exactly what being deaf means. Not understanding what it means risks my productivity and personal happiness. Being a programmer is my current profession, so there will be concrete examples about how being deaf affects me professionally as well as personally.

Solitude

I've been with 1000memories for almost a year, and I leave in two weeks to attend Hacker School. I still feel lonely sometimes. If your reflex is "everyone feels lonely sometimes," you would be right. But you would also be understating the loneliness we feel.

Deafness means I don't understand anyone. When someone talks at lunch, I want to know what they say. I miss out on the daily conversation, the back-and-forth, the friendships made after propinquity. And the worst part is that I don't have a choice in the matter.

Five years ago, I received a cochlear implant: a tiny technological machine implanted into my cochlea that fires electric bursts to help me hear. I had to learn sound all over again. I almost didn't qualify for the cochlear implant operation because, even at 16, I was considered too old. Teaching a child language gets exponentially harder as they grow. It's the same with hearing. I still can't tell the difference between "b" and "g," among many others. I might never, but there's no point in not trying.

In the past few months, I've felt like I'm the last person to know about things. I'm constantly surprised when something happened or changed. Once, an engineer left to work from Seattle the same week the two other engineers on the team left to present at RailsConf 2012. When I discovered that I would be the only engineer in the office the entire week, it was after everyone else had all gone.

It seems like a solution is just to ask more questions. I knew the engineer who was presenting at RailsConf Wednesday, but maybe I should have asked who else was going and how long he'd be gone? Maybe. I need to work on getting these questions to occur to me. It's hard when I still don't know these people very well, and haven't learned the social norms because I've never heard them.

Another solution is to somehow know what everyone else is doing. The engineers use Google docs to store priorities and to-do lists. We started using Yammer to keep the team up-to-date. That one isn't working too well — we get an email about every two weeks by a cofounder to use Yammer more. But the idea is to keep everyone up-to-date. Basically, company-wide toilet tweeting.

Group Conversations

In an open office like this, it's very easy to drop in on a conversation and add something. But without understanding what people say, the chances of doing that drop to zero. This is especially problematic in company meetings. The only way I can participate is with access services.

1000memories is still a startup, so we can't afford full-time access services. But for our most important meetings, the cofounders went through considerable expense to get me transcribers. With them, meetings are a bit better.

Since we work in an open office, parts of the team often chat with each other, especially at lunch. I always miss out on these talks, which are full of snippets of information no matter how bad the signal-to-noise ratio. This is really taken for granted. Any off-topic comment hints at an entire life to discover.

My best friend, then interning at Causes, kept telling me about random tips he picked up from other programmers because they were always chatting about new tricks they learned. This is how I learned about `git log -S`.

Have I told anyone about these problems? Have I taught anyone how to communicate with me? Yes to both. The result is no one uses the communication method after the initial novelty. The excuse my coworkers use for not practicing is "I need to practice." And then they keep on talking to you online rather than real life because it's easier and more familiar. That's because we're human, really.

I should have made it clear that Cued Speech was important to me. Startups are busy as bees, and people have other priorities. And I suspect I taught them too far down the line, because it was two months before I would be gone. Something I'll take away from this experience.

In a chat with a cofounder, I told him that I felt like I didn't have friends. I became jealous whenever a coworker talked to another and not me. It felt like a girlfriend talking to another guy. When they laugh and I'm unable to understand why, it feels like a punch in the gut, a giant inside joke I'm not part of. Maybe I should ask to explain the joke, even though most of the humor would be lost, because at least I would know.

I participate in conversations less than the quietest person I've met — not by choice, entirely. You should never want to be average — unless you are below average. This is a cry for normalcy, when so many others wish to be abnormal.

Love

Being deaf especially sucks when it comes to love. You can't ever love someone unless you've talked to them. So how do you communicate effectively? Everything I've ever thought of is awkward, because none of them are ever normal. Social norms are norms because they are what people expect.

I've talked to people "normally." It's hard, it's error-prone, and we have to repeat a lot. That's never a good recipe for love. It's hard to have awesome conversations when you have to repeat every other thing you say and are never sure whether the other person understood.

I could use an online dating site, like OKCupid. However, these are self-selected pools of people. There is a specific audience that goes to each site, and you still have to learn subtle communication skills which I currently lack.

I could do many other things (and am!) — exercise, dress well, maximize exposure. But in the end, I'm deaf. The most important thing is that I find someone who communicates well with me.

Job Interviews

Let me tell you a story in the present tense.

This morning, I get my fifth or sixth email from a large company in Washington. The second recruiter's trying to clarify some things, and she tells me that she'll be looking for an "interrupter" [sic] for my over-the-phone interview.

I ask to clarify this point. Since I'm deaf, having an interpreter for a phone interview wouldn't be very useful since the interpreter would be in Washington and I'm not, right? Would they actually be hiring an interpreter in San Francisco? How was this going to work?

In her next email, the recruiter delegates me to her manager, the third recruiter. At the end of the email is a copy-pasted message to be sure to fill out the necessary application for the interview. All interns have seen this application. I do not want to fill it out if they botch the interpreter.

Her manager tells me that she'll contact my college for access services and that we'll be using Live Meeting for the interview. She'll even do a test with me.

Unfortunately I have no idea what Live Meeting is. A quick search on Google tells me Live Meeting is basically Skype, but with no clues on how to download it.

I end up never starting the interview.

Accessibility in Interview Applications

For most company interview applications, they ask for a phone number without alternatives. I put a random note where I can. Something like "Since I'm deaf, I can't do a phone interview, but you can reach me at…" at the end of the "Why do you want to work for us?" question. I never hear back from these. I don't know if it's whether they never saw my note, whether they rejected my resume silently, or whether they attempted calling my phone number (which doesn't take calls).

Last time I applied, even Google didn't provide alternatives. It was a beta application, though when I emailed a recruiter in charge of my college about it, she was very helpful and understanding.

HR is behind the times. There is no reason interviewing over the phone is better than interviewing on video with *typewith.me*, Skype, or Gmail chat. I've done all of these, and it's always turned out that the programmers preferred to conduct those interviews this way.

Screencasts, Talks, and Video Tutorials

When I was trying to learn Rails, I soon found out that a large chunk of popular tutorials were uncaptioned screencasts or videos — a huge body of knowledge I'm unable to tap into. Even Khan Academy went uncaptioned until recently when an independent group helped out. So many uncaptioned videos exist because minorities are not prioritized.

Since I can't listen to talks, I have to make-do with slides. Slides almost never go into the depth a talk does: it's all surface knowledge.

In the end, I learned programming by a combination of getting lucky, enrolling in formal classes, poring over books, Googling, finding Stack Overflow, and making things, making things, making things.

Access Services

There are three layers involved in translation: the messenger herself, the interpreter, and me. I hear what someone says through the lens of someone who probably doesn't know programming.

As university subjects get harder, access services get worse. In Probability class a year ago, we learned about second derivatives and gamma probability functions, and the typist that my college hired for this class was typing a transcription indistinguishable from a novel written by a chimpanzee. Typists are not required to learn the prerequisites, nor do they have to learn

along with the student. They just type what they think they hear.

It's like playing a game of Telephone — the classic example of lousy communication. Which means it is never, ever as good.

The only good access service I've ever gotten is Cued Speech. In a basic sense, Cued Speech is a system that uses signs for sound. It was invented to battle the spectacularly low deaf literacy rate. (The average reading level of deaf 17- and 18-year-olds is at the fourth grade level.) With Cued Speech, I see exactly what the messenger says, without ambiguity. The only error arises when the transliterator mishears the person. Unfortunately, in college, I'm not offered Cued Speech due to politics not worth mentioning, and Cued Speech is not widespread.

Deaf Culture

I never considered myself part of Deaf culture. It arose because, I suspect, we were lonely. It's the same for any minority. Except this time, Deaf culture came together because of a common language everyone could understand — American Sign Language. I've heard the stories. Deaf people entering college for the first time. Finding other deaf students. Suddenly, during their first sleepless night, they're making up for all the conversations they had missed.

Some become angry at the hearing world. They went so long without feeling like they belonged. Without feeling loved.

Some don't think deafness is a disability; it's just a way of life. After all, we can do anything except hear. But I don't want to be part of the Deaf world, which seems so cloistered sometimes. I want to be part of the larger world — and out here, not being able to hear is a pretty significant disadvantage.

Friends

Despite the constant communication barriers, I've really grown into a good programmer during my time at 1000memories. I've learned how to communicate with others, what the real world is like, how to do behavior-driven development, mastered JavaScript, and even submitted a patch for Ruby on Rails. They gave me a chance prove myself as a programmer — and, in these past few months, as a friend.

Nine months had passed since my inception as a bumbling intern before I admitted to a cofounder that I was feeling lonely. It happened after one social bowling night, when a scheduling mistake caused us to wait in the alley for an hour and a half chatting in a noisy environment. I stood off to the side, feeling stupid, watching my coworkers laugh. I didn't want to see that. As soon as I collapsed back in my apartment, I cried. Then a little thought went off in my head: shouldn't someone know about this? So I wrote an email.

The next morning the cofounder read my email. He invited me for a chat over breakfast. When he let me into his apartment, I was surprised and a little guilty when I saw his eyes. That moment was the most vulnerable I ever saw him. Of course, you idiot, I thought right then. Founders get lonely, too.

We had bagels on his couch. It took a long time before either of us started talking. I began with what my life used to be three years ago, when I was a completely different person. I was so passive and shy I couldn't look anyone in the eye. I blamed everything. I depended on everyone. I was content to live life as a cog in an industrial machine. I just wanted an easy life and die of old age. Until I went to college.

In college, I was depressed and bored. I felt like I was missing something. Then someone I knew died. And another. I realized I wasn't missing anything. Happiness is a verb. And now, me-three-years-ago wouldn't be able to recognize who I am today.

The cofounder and I went through what we could do to make my life at work better. Part of my contract involved a budget for speech therapy, something I never took advantage of. I brought up teaching Cued Speech. He mentioned that, at lunch, I could nudge someone and ask what we were talking about. We talked about going back to college, living in the adult world, and finding love.

I walked away feeling like we could be friends. ■

David Peter is a hacker who likes drawing, writing, and reading fantasy/sci-fi. He was the frontend engineer at 1000memories, and is now attending Hacker School in NYC before he returns to college in the fall.

Reprinted with permission of the original author. First appeared in *hn.my/deaf* (davidpeter.me)

Photo taken by Shirmung Bielefeld.

Body Hacking: My Magnetic Implant

I've got a magnet implanted in my finger. Here's my story.

By DANN BERG

Let's talk about magnet implants. I don't really bring it up much, but I have a small rare earth magnet implanted in the pinkie finger on my right hand. I've had it for around three years now.

Whenever someone finds out that I have this implant, they've always got a ton of questions. Usually the first question is "why?" While this is a valid question, I tend to dismiss it when asked, favoring a continuation of the conversation rather than a Q&A session about my motivations and the way I view and interact with the world. But to be fair to those who are genuinely curious (rather than those who quickly ask "why?" out of shock), I figured I'd share some of my thoughts and experiences related to having a magnet implant and hopefully answer some of the frequently asked questions.

Initial Interest

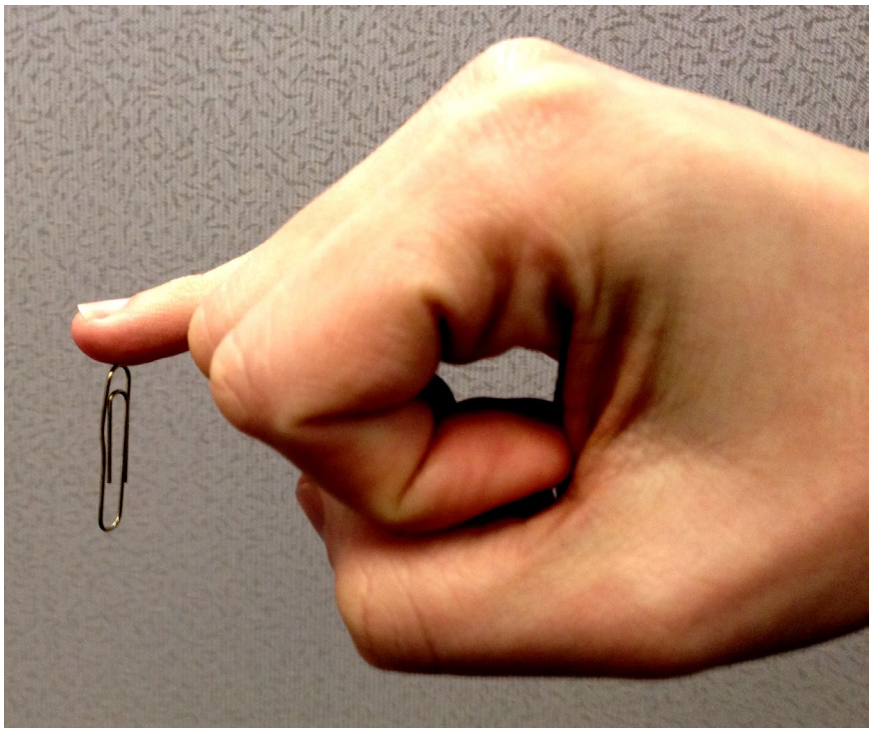
When I first read about magnet implants, the technology was still in its infancy. There was an amazing article by former editor and founder of BMEZine Shannon Larratt that discussed a type of "sixth sense" that these magnets provided. But my interest in getting the procedure myself was quickly squashed after reading and seeing, in graphic detail, what happened when the silicone encasing around these magnets broke down and the magnet corroded inside his finger. Even after all these years, just the thought of those images gives me the chills.

At that time, I wrote the procedure off as a cool concept and nothing more. It wasn't until around three years ago, in a bar with some friends, that I learned that one of my buddies actually had a magnet implant. Not only did he have one, but he had had it for a year at that point. My interest was immediately piqued again. No longer was this just some procedure that someone tried and failed. Rather, it had a bit of staying power. I spent the rest of the night talking to him about the implant and watching him perform party tricks, such as picking up paperclips and bottle caps with the tip of his finger.

That night, I made the decision to get the implant. The next day, I contacted my local experienced implant practitioner and made an appointment.

The Procedure

The first decision I needed to make was which finger to get the implant. I had already chosen to get the implant on my right hand, as I am left handed. I ended up settling on my pinkie finger after performing a bunch of routine tasks and paying attention to which fingers I used most. Another common finger for the implant is the ring finger, but I felt like I used my pinkie less, so I opted for that one.



The actual implant procedure was fairly quick. My finger was marked in two places: where the magnet was going to go as well as the incision spot (around a quarter to a half inch away from the final resting spot for the magnet). He then made the incision with a scalpel, used some tissue elevators to separate the tissue, slid the magnet into place, and sealed the incision with some surgical glue. Next was a bit of tissue compression; then my finger was wrapped up and I was on my way. It took about fifteen to twenty minutes total.

Initial Reaction

Due to the tissue trauma from cutting open my finger, it took a while before I could really take full advantage of having the magnet implant. It was only a day or two before I picked up my first paperclip, but it took a few months before my finger really regained full sensation. The tissue around the implant all the way up to the incision point was swollen and fairly numb for weeks after the procedure. This didn't stop me from playing with the magnet all the time, picking up paper clips and other small metal objects.

As the swelling went down and the sensation in my finger tip came back, I began to experience elements of an invisible world around me. When people discuss magnet implants giving a "sixth sense," this is what they're talking about. I was working retail at the time, and I believe the first thing I noticed was the vibrations from the fan inside the cash register. I could feel the invisible field, coming out of the side of the computer in a half-dome. The vibrations varied in strength depending on where I held my finger. It really did not feel like a foreign object vibrating, but rather my finger itself. It was an extremely weird sensation and fairly uncomfortable at first.

Another uncomfortable experience, which I quickly learned to avoid, was handling other magnets in such a way that they flipped the magnet inside my finger. The magnet inside my finger is round and flat, so introducing an outside magnet with a different polar pull would cause my magnet to make a quick flip inside my finger. While this didn't hurt, it was (and still is) fairly uncomfortable. In addition, sometimes the magnet would get pulled on its side, sticking up and down rather than settling flat in my finger. This never hurt either, but also proved to be quite uncomfortable and required a quick massage to get the implant to lay flat again.

Experiencing the World

I quickly learned that magnetic surfaces provided almost no sensation at all. Rather, it was movement that caused my finger to perk up. Things like power cord transformers, microwaves, and laptop fans became interactive in a whole new way. Each object has its own unique field, with different strength and "texture." I started holding my finger over almost everything that I could, getting a feeling for each object's invisible reach.

Portable electronics proved to be an experience as well. There were two fairly large electronic items that hit the shelves around the same time as I got my implant: the first iPad and the Kindle 2. Both of these items had a speaker located at the bottom right of the unit, almost exactly where I rested the pinkie finger of my right hand. Both of these speaker magnets were powerful enough to flip the magnet in my finger if I brushed past them in a certain way. This was incredibly annoying, but became a moot issue

as soon as I put a case on either device. No elements of the iPhone ever posed an issue, and newer versions of the Kindle and iPad moved this magnet to a non-intrusive location. I was very wary of the iPad 2′ s magnetic smart cover, but these magnets are so specifically targeted that they take a while to find even if you're looking for them.

The best part of having the magnet implant was discovering invisible magnetic fields when I wasn't actually looking. The first experience I had with this was walking through the intersection of Broadway and Bleecker in Manhattan. I passed through this intersection a few times before realizing that my finger would tingle at a certain spot. After paying a bit more attention, I realized that I was feeling something underground. At first, I assumed it was a subway car, but later came to the conclusion that it was most likely the subway power generator, or the giant fan that was cooling these generators. After noticing these underground waves at Broadway and Bleecker, I began feeling them all over Manhattan.

Another unexpected magnetic field is at certain store checkout counters. Mainly at book stores (and some clothing stores), there is a strong unit under the counter that removes the security tags. This unit usually pulses, sending out magnetic waves strong enough to be felt a few feet away. This always leads to interesting conversations with the cashiers.

Downsides

There are surprisingly few downsides I've experienced in the three years that I've had the magnet implant. Luckily, the magnet is not strong enough to wipe out credit cards nor will it negatively affect electronics or computer monitors. I've also flown numerous times since having the procedure done and never had any issues.

The only real negative aspect to having this implant is the inability to get an MRI (if needed) without first having the implant removed. This is something that I thought about before getting the procedure done, and I made a conscious decision to get the implant anyway. I also figure that if I'm ever incapacitated and put in an MRI machine without the ability to give the doctor any forewarning, a tiny magnet getting ripped out of my finger will be the least of my concerns.

Thoughts Three Years Later

Three years after getting the implant, my magnet is something that I constantly forget about. It's not something that tends to come up in general conversation. Even the prompt, "Tell me something unique about yourself" often occurs in an environment where mentioning a magnet implant may be slightly inappropriate. As far as my personal use of the magnet, it serves as more of a general curiosity tool rather than having any sort of practical use. I'm not in a profession that requires me to tell live wires from dead wires. Rather, if I find an object that's labeled "magnetic," I'll generally hold my finger up to see the exact strength of the magnet and nothing more.

Over the years, the magnet has lost strength as well. While I could once hold a large paperclip, the magnet now only supports a small one. The combination of a weaker magnet and the novelty wearing off means I rarely even think about the implant. It's only when I sense a fairly strong field that the magnet will enter my consciousness, and even then, it's usually a quick mental note before I continue doing what I was doing.

Despite this, I'm still really happy that I had this procedure done. It has unlocked an entirely new world for me, one that I can touch and interact with in a very real way. While a magnet implant doesn't technically count as a "sixth sense" (it's more of an extension of our existing sense of touch), the way that the body internalizes these tiny magnetic vibrations feels truly foreign.

If this is something that you're interested in getting done, by all means continue your search. Make sure to do your research, find an experienced practitioner, and know the supplier of the magnet. Understand the risks and the consequences of getting a magnet implant. Once you've done all get, go get the magnet and start exploring the world. ■

Dann Berg makes stuff. He is the co-founder of hangalong and a writer at LAPTOP Magazine. Say hi to him at @DannBerg

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

MEMSET® HOSTING

Rent your IT infrastructure from Memset and discover the incredible benefits of cloud computing.

MINISERVER™ CLOUD COMPUTE

From £0.015p/hour
to 4 x 2.9 GHz Xeon cores
31 GBytes RAM
2.5TB RAID(1) disk

MEMSTORE™ CLOUD STORAGE

£0.07p/GByte/month or less
99.999999% object durability
99.995% availability guarantee
RESTful API, FTP/SFTP and CDN Service

SCAN THE CODE FOR MORE INFORMATION

MEMSET® HOSTING

CarbonNeutral® hosting

ISO 9001: Quality
ISO 14001: Environmental
ISO 27001: Security

PC PRO Excellence AWARDS 2011 WINNER BEST WEB HOST

Best Managed Service WINNER The ISPAs 2011 www.ispaawards.org.uk

Find out more about us at www.memset.com

or chat to our sales team on 0800 634 9270.