

MicroISV on a Shoestring

B2C stands for "Bingo To Customer"

- [Home](#)
- [Start Here If You're New](#)
- [Greatest Hits](#)
- [About This Blog](#)
- [Subscribe](#)

Getting A New Product Off The Ground: Part Two

By [Patrick](#) on

Along with some other users from Hacker News, I'm doing my darndest to get [Appointment Reminder](#) into the hands of customers by the end of November. It looks like it is going to happen, too. (There was an early blog post about this [here](#).)

Rails Deployment Options

Somebody asked me to talk about this subject, so I will cover it briefly. Rails deployment has historically been very painful. These days it is a little better. The big options are:

- Run a Rails stack on a server or VPS under your control
- Use Heroku
- Run a Rails stack on Amazon EC2

I have used Heroku for a client project before. It is a wonderful service, don't get me wrong, but I have almost five years of managing production Rails applications on a VPS and less than five weeks of managing production Rails applications on Heroku. I know from experience that I will try to do things during development where Heroku's limitations (like the read only filesystem) will trip me up, and tripping myself up doesn't get things in the hands of my customers any quicker.

Additionally, my anticipated traffic for Appointment Reminder is way, waaaaay within the capacity of single largish Slicehost slice to handle. If I manage to bring my Slice to its knees, it will be because my revenue has hit several million dollars a month. That will give me many attractive scalability options, such as hiring someone to care about scalability while I sip chilled juices on the beach of a tropical island.

I built a staging server, from scratch, and obsessively documented the process. (I strongly recommend that you do this for building servers, since you will have a burning need for that documentation if something ever goes wrong.) It relied heavily on the [Deprec gem](#), which is far and away the easiest

way to get a Rails stack running on a bare Ubuntu install. I still spent almost 4 hours filing down sharp edges, though. (Sample: I wanted to run Rails 2.3.10 rather than upgrade to Rails 3, there were issues with compiling Mongrel on Ubuntu that had to be resolved via manually grabbing packages, etc etc.) I only really recommend this if you know what you're doing with Rails system administration.

In terms of software choices for my deployment stack:

- Ruby 1.8.7 via the Matz Ruby interpreter. It isn't the fastest, but speed is hardly of the essence to this application and it is the least likely to die horribly when using a gem or library, since everybody tests against it.
- Rails 2.3.10. I don't want to learn Rails 3 at the moment. Maybe next year
- Mongrel cluster: I have experience managing it.
- Nginx: The best web server I've ever had the pleasure of working with. Also plays well with PHP, which I need on the box to handle the marketing portion of the website outside of the application. (The marketing site is in WordPress.)
- God process monitoring: I have experience using it.
- DelayedJob job queuing: I have experience using it.

I will probably eventually put WordPress on a physically separate server (hey, it's WordPress), but that doesn't strike me as the biggest priority at launch. I did go to a wee bit of trouble to secure it:

- Access to the admin directory is denied at the server level for requests not from my private network.
- WordPress has its own database and database user, and can't touch anything elsewhere.
- The WordPress directories are only writable by root, not by the web server user. If I want to install plugins, I get to SCP them to my account on the server then SSH in and start sudoing to actually copy them in. Ditto for uploading files. Facilities to upload and immediately execute PHP code are, ahem, a persistent source of vulnerabilities.

Code Is About 70% Complete

I have sustained pretty close to my lifetime peak productivity at programming (say that five times fast) for the last two weeks, averaging 4 to 6 hours per weekday. Appointment Reminder has most core functionality implemented on the staging server, for a single single-user account. [Twilio](#) has been an absolute joy to work with — I really can't express how much of the pain they take out of running a telephony company.

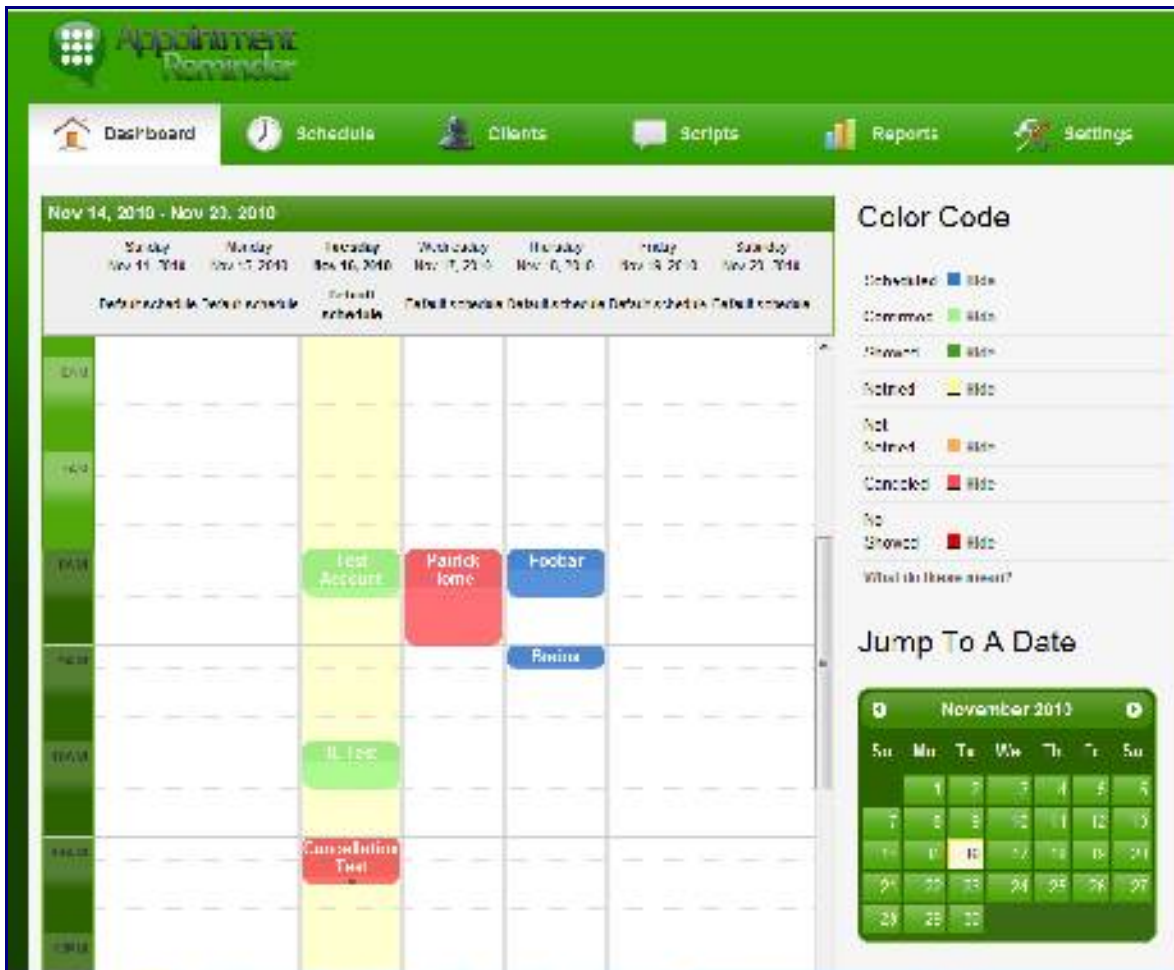
What's done:

- Client creation/management: 100%
- Appointment creation/management: 90% (still need to associate scripts with appointments)
- SMS reminder loop: 100% implemented. (web app -> Twilio -> user's phone -> web app)
- Phone reminder loop: 100% implemented (web app -> Twilio -> user's phone -> web app)
- Dashboard: 90% implemented. (Needs to be a wee bit more configurable, particularly for multi-user environments.)

Things are looking fairly slick thanks to jQuery UI and a lot of AJAX, but it has been a bit of a handful to test. If you have suggestions for doing so, I'd love to hear about them in the comments. Mostly I have been doing unit testing for models which are likely to have failures, and then doing manual testing to verify that the UI works how I expect it to.

I am currently using an absolute riot of colors on the dashboard to convey status information. That

probably should get reduced. SASS has been very, very helpful in keeping me from having to handwrite CSS to manage all of these.



What still needs work:

- Email reminder loop: 0%, should take ~4 hours for simplest thing that will work
- Script creation/management: 0%, should take 1~2 days (this requires hooking up another phone system)
- User/account management: 0%, should take ~2 days
- Reports: 0%, should take ~1 day for simplest thing that will work
- Feedback to user on appointment status changing: 0%, 2~4 hours depending on how full-featured I want to launch with
- Charging people money (using [Spreadly](#) — shouldn't take more than a day)

Outsourcing Up A Storm

I have had a lot of luck using [Fiverr](#) for getting recordings done cheaply. Try the [Appointment Reminder demo](#) to hear one of the voiceover artists in action. My plan is to launch with about four different sets of voices to give customers some variety: I expect that most will actually record their own reminder scripts, but anything which decreases the amount of work it takes before they get a phone call from the system is a win for me.

I also plan on eventually using reminder scripts as a cheap way of marketing. There are virtually

infinite marketing angles once you have the ability to make a phone ring, since it is so ridiculously compelling. (I mean, people paid how many *billions of dollars* for ringtones?) For example, I have an absolutely unhealthy interest in the [Old Spice Man](#). Getting a script inspired by the character is easy, remarkable, and is both a win for my customers (“That phone call you gave me earlier was hilarious! Thanks!”) and lets my customers market AR inside their organizations (“Cindy, you have to hear this. Quick, what is your phone number?”)

There are virtually infinite variations on that: you could imagine getting a phone call from a Humphrey Bogart impersonator, etc etc.

Speaking of outsourcing, I recently started using a Virtual Assistant, something I have been meaning to try since reading about them in Rob Walling’s [book](#). I found mine through an agency called [Pepper](#) (that name is a bit of an easter egg for comic book geeks in the audience). For about \$300 a month, I get twenty hours of time from one particular lady who works at their office. She does, within reason, anything I tell her to do. This has been a huge win for me in cleaning boring, time-sucking tasks off my plate so that I can concentrate on development and marketing.

For example, up until quite recently I was ten months behind on bookkeeping. That’s a long story: I do expenses bookkeeping via a home grown application so that I can display [nice graphs](#) on my website, and that application got broken by an unrelated update back in February. (Bookkeeping of revenue is totally automated, and didn’t break.) Since customers don’t pay me to do bookkeeping, fixing it went on the back burner. I eventually got it working again sometime in summer... and then had six months of credit card statements to process. Well, that sure sounds like *work*, so I procrastinated... until there were ten months of statements to process. Crikey. It kept getting worse, it kept weighing on me that it wasn’t done, and I didn’t want to spend a day or two just going through 20 PDF files and typing some 300 transactions into the computer.

Enter my Virtual Assistant. I spent ten minutes typing up my decision tree for classifying transactions (the credit card statements have mixed business and personal charges — whoops, got to fix that one next year), instructions for operating the bookkeeping interface, and the like. Then I created an account for her in the system, zipped up the 20 statements, sent them over, and answered an email or two from her over the course of the next week. Managing her and reviewing her output took perhaps 30 minutes in total, instead of somewhere on the order of three to six hours to do the data entry myself — plus, it didn’t totally wreck me for other productive work that day.

I’m now almost caught up with bookkeeping (got another statement delivered by my bank in the interim), and now that there is a process in place for it keeping caught up is simplicity itself: download statement, shoot it to my VA, go on to do things that actually matter.

This was **astoundingly** beneficial for me. I have a nagging worry taken off my plate and a process to make sure it never comes up again. I got to invest several extra hours in things which matter to the business, instead of doing data entry. That covers the first week of the month: if I never got another stitch of work done, I’d consider my \$300 well spent, but I’m guessing that I can almost certainly find other stuff for her to do.

When I get some time to breathe this month, I’m going to figure out other tasks I can assign to her: perhaps drawing up lists of keywords or blog post topics. I mean, I could spend a day coming up with every possible permutation of a professional service plus every possible synonym for “scheduling software”, but that doesn’t require my personal attention. (Why I would want a list of hundreds of keywords about my software is fairly straightforward if you’ve followed my bingo card creation endeavors: implement system to create content at scale, farm out content creation to freelancers, cover the organic search longtail for the topic, sell thousands of accounts to organic searchers).

Speaking To Customers

Dharmesh Shah, I think, said that there are phone people and email people, and that most engineers are email people. I am definitely an email person. I hated speaking on the phone even before I started using the Internet. But I've been chitchatting with prospects (at Ungodly O'Clock AM) for the last couple of weeks to hear about their needs, and have both learned stuff I didn't know about my market, confirmed some stuff I did know, and — importantly — gotten some promises to pay me money **immediately** when the software is ready.

Here is my mental model of the typical customer for Appointment Reminder: Martha owns a massage therapy practice. If Cindy, her client, doesn't make her 4:00 PM massage, then Martha loses \$60 of revenue immediately and will likely be unable to rebook the slot, since she'll find out about the gap at about 4:05 PM. Appointment Reminder either gets Cindy in on time or gets Martha 24 hours of notice, so she can book the slot. Martha happily buys the service.

Note the embedded assumption there: Appointments are something you *go to*. I never even realized I was assuming that. It turns out that I was, and that there is a broader market than I expected, for appointments where the service provider comes to you.

Why does this distinction matter? Because Martha is greatly annoyed when her customers don't come in, but she has not literally lost money out of her pocket.

Consider Earnest the Exterminator. He has a team of three guys and a van filled with lots of dangerous chemicals. When Earnest makes an appointment, he finds out that you've forgotten **after he drives 25 miles to your house**. Not only did Earnest just lose out on the revenue from killing your bugs, he just burned up **three hours of salary** at skilled labor rates because you forgot that today was, indeed, Tuesday. Martha is greatly inconvenienced by her no-show problem. Earnest is **sputtering with rage** about his now-show problem. I now know this because I've gotten on the phone with three different Earnests now and just *shut up* while they talked about their problems.

I'm going to make changes to my marketing posture to reflect this (the Marthas of the world are overwhelmingly female, but there are an awful lot of guys in the Earnest category along with some ladies), and I've also started to create some features for supporting Earnest and his team. Some of my Earnests have been surprisingly tech savvy ("Oh, my programming team wants to know about your API." "You have a programming team?!?" "Bleep yes I do, you think I'm going to run the systems myself?"), and the knowledge that the user of the software could very possibly be mobile rather than at her place of business really rejiggers my development priorities for e.g. checking your schedule via a phone call.

Plus, as you can probably guess, somebody who has a programming team and a very good idea for how many tens of thousands of dollars they lost last year to no-shows is, shall we say, quite willing and able to pay any price I want to charge.

Next Steps

More of the same, really. I just got confirmation from one of my freelancers that she is working on the MP3 files for phone calls. Tomorrow I build out the script interface to allow people to pick or record the telephone/SMS/email scripts that they use. After that is done, I think I'll do the email integration, which is the last bit of actual functionality for the site. After that, user management (if worse comes to worse, all I really need is to have users have separate settings, and I can do all creation of secondary users by waiting for emails from customers and then doing it via the Rails console rather than with a slick UI on top of it).

If you have anything in particular you'd like me to cover in the next installment, say the word.

This blog is about the business aspects of running [Bingo Card Creator](#), a small software company. Want more great articles? I keep a list of [my best work](#) curated. A brief summary of the last few years is available [here](#). If you like what you see, I encourage you to sign up for the [RSS feed](#). Thanks for visiting!

Posted in [Appointment Reminder](#) | [14 Responses](#)

Getting A New Product Off The Ground: Part One

By [Patrick](#) on

There was overwhelming enthusiasm from people when I offered to blog about the development of [Appointment Reminder](#), so I will be doing it during the month of November, prior to my planned release. (Tentatively planned for the end of the month, if it is ready in time.)

Achieving Activation Energy

Appointment Reminder has, theoretically speaking, been on my plate since April 1st, which was my first day of self-improvement. I released the MVP — basically, a functioning demo of the core interaction between service provider and customers — halfway through May. And then I sat on my hands for about six months.

Oh, stuff happened, don't get me wrong. I went travelling internationally, twice. I kicked butt and took names for some consulting clients, got (and turned down) about a dozen job offers, won an award for best presentation at the Business of Software conference, started writing an article for the ACM, met a young lady, and broke my old bingo card sales record. But while my life is firing on all cylinders, happiness does not write jQuery or Rails code.

My last client project wrapped up in mid-October, I told clients I would be mostly unavailable for the next few months, and I picked the end of November as an arbitrary deadline to finally get Appointment Reminder out the door. Deadlines tend to coerce me into doing my best work. Specifically, deadlines with subgoals that have small units of measurable accomplishment work best for me. It must be the WoW player in me, I swear. So I broke the month or so of work up into a series of mini-quests which take about half a day to accomplish, and then logged the first dozen or so in [EpicWin](#) (productivity management for recovering WoW players — my other quests include going to the gym, doing the dishes, and calling each of my younger brothers). I have been polishing them off more or less according to plan.

Technology Choices

When I start a greenfield project (and AR is still new enough to be mostly greenfield), one of the first things I write down in the project notebook is what technology stacks I'm good with and which make a fit for the product. Given that my options for web development are Big Freaking Enterprise Java and Rails, Rails was the clear winner. I am literally an order of magnitude more productive in Rails than I am with Java, and I lack the experience with Java architecture astronomy to build an application from the ground up (which I have done successfully in Rails before).

I sometimes also use the opportunity of new projects as an excuse to broaden my horizons. For

example, I've wanted to get into jQuery for a while since the world seems to be moving that way (away from Prototype, my old Javascript framework of choice), so I decided to take the minor productivity hit with starting a new framework and moved to jQuery.

My other professional growth goal with AR was to get a little more serious about interface design.

Thomas Ptacek has been raving to me about how much better SASS/Compass made the experience of getting CSS to work right, so I decided to move to SASS/HAML from my previous CSS/ERB standbys. This delivered huge, huge wins within two days of starting exploratory coding with the project. I can't recommend [SASS](#) enough. This also led to a bonus: the lack of markup going into my views means that I can develop against a cheapo CSS template and then swap to a professional design later, without having to have the design be on the critical path for November.

Design on Paper

I generally keep all notes for a project in a \$1 notebook. However, my first Appointment Reminder notebook is nowhere to be found. Drats. I probably left it in a hotel room in America. So I bought a new notebook and started sketching out database tables, screens, features I would like to include, etc etc. Then I started cutting, ruthlessly.

For example: Users need to be able to manage clients. OK. Hypothetically, they may have hundreds or thousands of them, so they'll need a search interface... but will they have hundreds or thousands on launch day? No? *Search is out of scope*. Next!

Users need to be able to give custom reminders to clients. This requires recording their voice.

Uploading files is a pain in the keister — *out of scope!* I'll make do by having them just narrate the reminder over the phone to Twilio, which would obviate the need for me to do any file encoding or management myself.

And so it continued. By the end of my first working lunch, the feature list that was in scope ended up looking something like this (broken down roughly along controller/model lines):

Accounts

Account creation

User management

User permissions (Enterprise-y feature, enterprises aren't going to adopt in first two weeks: out of scope.)

User preferences

Login / Logout / Forgot Password

Clients

CRUD app for users to add clients

Track phone numbers and email

Communication preferences

Opt out of reminders in case of abuse (Barely avoided being out of scope: it is important but doesn't sell software.)

Schedules (one schedule manages one resource — a room, service provider, etc)

Default schedule (most users will have only one)

CRUD app for schedules

Change hours of business day, days of business week. Eventually that might change on week-to-week basis, for now, simplest thing that works.

Appointments

Appointment calendar — starred several times because this is going to be the hardest part of the app

CRUD for appointments, based on calendar

Status tracking for appointments

Reports for number of appointments missed, etc Can't be used on day one, out of scope.

cron job to send reminders about appointments

Reminders

Spawned automatically from appointments

State machine (see below)

Twilio integration

Twilio

Inbound calls

“Who the heck are you and why did you call me?” auto-response

Reset trial (already implemented)

Record custom reminder

Get schedule for day dictated over phone great feature, out of scope for now

Voice-assisted tour only in scope if I have two days left over

Outbound calls

Reminders

If answered live, capture user input and update Appointment state accordingly (coming, canceled, needs call, etc)

If answered by machine, leave message, update Appointment state as notified.

Reminders with custom recordings

Text messages

Email

Thank you for signing up, blah blah blah

Password reset

Appointment reminders

User notifications on appointment confirmation, cancellation, etc out of scope

Billing

Get paid (oh heck yes, in scope)

Assorted Stuff

Account deletion out of scope

Admin interface God gave us console for a reason

Custom analytics Nobody to track on first day

A/B tests (would be out of scope, but hey, free since I have [A/Bingo](#))

Whitelabel version

HIPPA compliant version

Change tracking (e.g. audit trail for deletion, etc)

Exploratory Coding Begins

After playing around with the signup screen for a little while, mostly to get a feel for SASS/Haml, I started working on the first feature of the app that would provide user value and be mostly self-contained. Ideally, I would start with the first thing I want them to do (schedule an appointment), but that requires adding a first client anyhow, so I started with the client administration screens first.

I don't use Rails scaffolding, although since I got publicly told off [invited to try improvements](#) by DHH for having insufficiently RESTful routes I thought I'd give that a try. (Turns out it actually does save enough pain to matter, although I still think publicly visible RESTful routes are a mistake. For internal features of a CRUD app, though, they make a lot of sense.)

What I generally do is start with the index action, verify that I can display records added to the database directly (via the console or a seed script), then start on the show action. This results in me getting *frequent incremental visual progress* that the program is, in fact, getting better. Anything which can't be used yet gets stubbed out with lorem ipsum. When I feel myself reaching the stopping point for a day, I go to the next action on my list and get the view working, even if it is as simple as displaying a screen which says "This action does not really exist yet but if it did it would show #{@client.name}".

That gives me a place to pick up again in the morning (or, ahem, afternoon, given my frequent work/sleep habits).

Anyhow, after getting show done, I work on the edit/save loop, which requires building out the model a bit, adding validations, and writing some less trivial controller logic. Here I frequently discover something like "Hmm, it would be handy to have a way to display messages out of the flash" and start working up a helper to do it in a minimalistic way. You can see it behind the example of me operating the new (error-enabled) edit window.



Working with jQuery slowed me down a bit when creating some of these screens. I used [jRails](#) to ease the transition (it lets you use all the old Rails-esque Javascript helpers like `link_to_remote`), but it does not play well with jQuery 1.4 out of the box. It turns out that the issue I was having was mostly that jQuery executes any Javascript it grabs from the server prior to updating the DOM with the rest of the data it got back, which borks the Prototype-esque habit of sending back both HTML and then Javascript which relies on that HTML being in the DOM to function. I got around this by simply returning only Javascript — e.g. for the “pop a window to put in a new client” action I return:

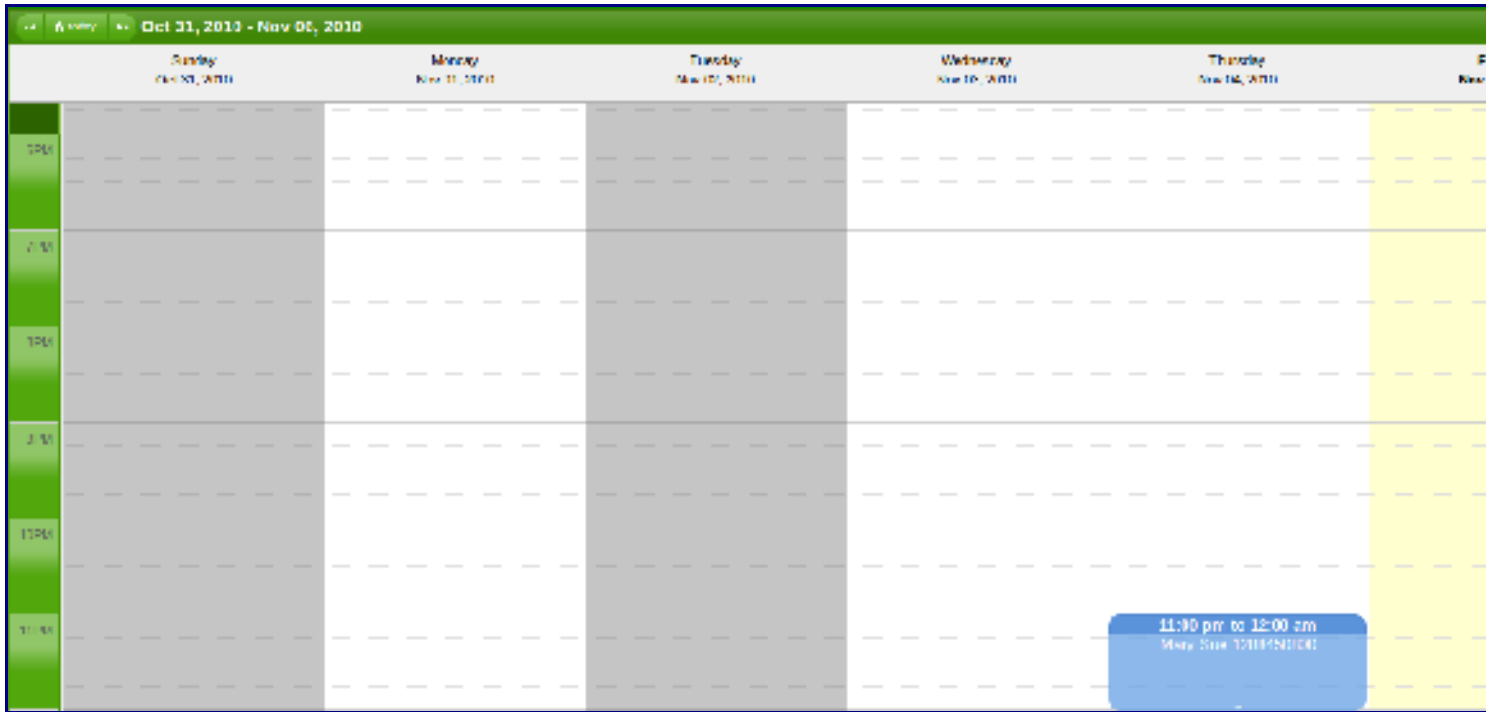
[view sourceprint?](#)

```
1. :javascript
2. $('#client_dialog').html("#{escape_javascript(render :partial
=> 'edit_form')}");
3. $('#client_dialog').dialog({
4.   //blah blah
5. });
6. $('#client_dialog').dialog('open');
7. $('#client_dialog').find('input').focus();
```

This is hacky as heck, and obviates some of the reason of using jRails in the first place, but it works. I can always DRY up the helpers a little more later.

After getting the validations and whatnot working for Clients, and doing some light testing to make sure things worked to my satisfaction, I did similar work for Schedules. I got the basic CRUD functions working in record time, mostly by copy/pasting everything I had done for clients and then just changing the parts of the model and view that were different. The controllers are mostly identical, at least until I actually create the ability to render a schedule.

I've finished the CRUD logic for my first two parts of the problem domain, and am impressively ahead of schedule. To celebrate, I spent a day upgrading jQuery to the latest stable version (not quite so fun), upgraded the jQuery week calendar widget I was using to a fork that is being actively maintained, and wrote code to render the minimum calendar possible without actually having any appointment data available.



And that is about it for today. Tomorrow: hooking up the ability to add appointments to the schedule, then CRUD logic for them, then the ability to create an appointment and client at the same time. That will complete the first take on the core interface logic for the application, leaving me next week to tackle integration with [Twilio](#). (I had originally planned on that taking at least two weeks, but I remember it being much easier than I expected back when doing the MVP, and I seem to be progressing faster on this project than I typically do.)

Plans after that are roughly:

- Turn on user account creation, log in/log out, etc
- Create staging server, taking care to document my setup process on a computer this time (darn missing notebook)
- Add integration with [Spreadly](#)/Paypal for billing
- Polish as much as possible.
- Test as much as possible.
- Ship at or near end of November.

After Shipping

After shipping, of course, comes the 90% of the remaining work needed to turn an application into a business. (Of course, I have been doing some of it all along: speaking with customers, gathering requirements, thinking about marketing angles, etc.) In particular, I'm starting to devote my free cycles at lunch into thinking about scalable content generation strategies for organic SEO, which is the form of marketing that I've had the best results with in the past.

But it is highly likely that, immediately after launch, I'll have a bare handful of customers and a fairly relaxing December with answering customer support queries (and pre-sales inquiries), mapping out future development, starting the marketing engine, and enjoying Christmas with my family. Then, on to January.

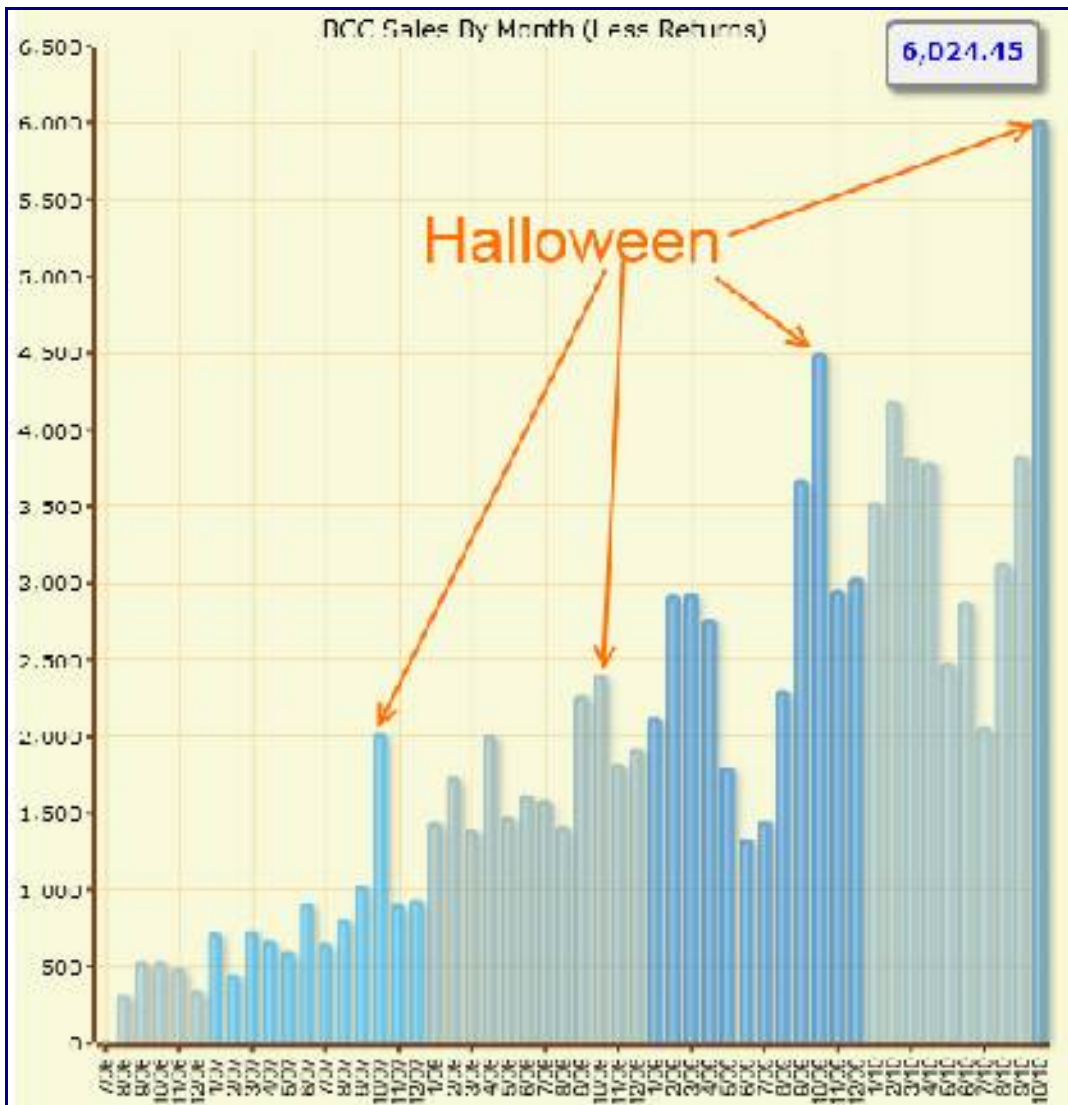
Posted in [Appointment Reminder](#) | [19 Responses](#)

How A Half-Broken Halloween Promotion Smashed Revenue Records

By [Patrick](#) on

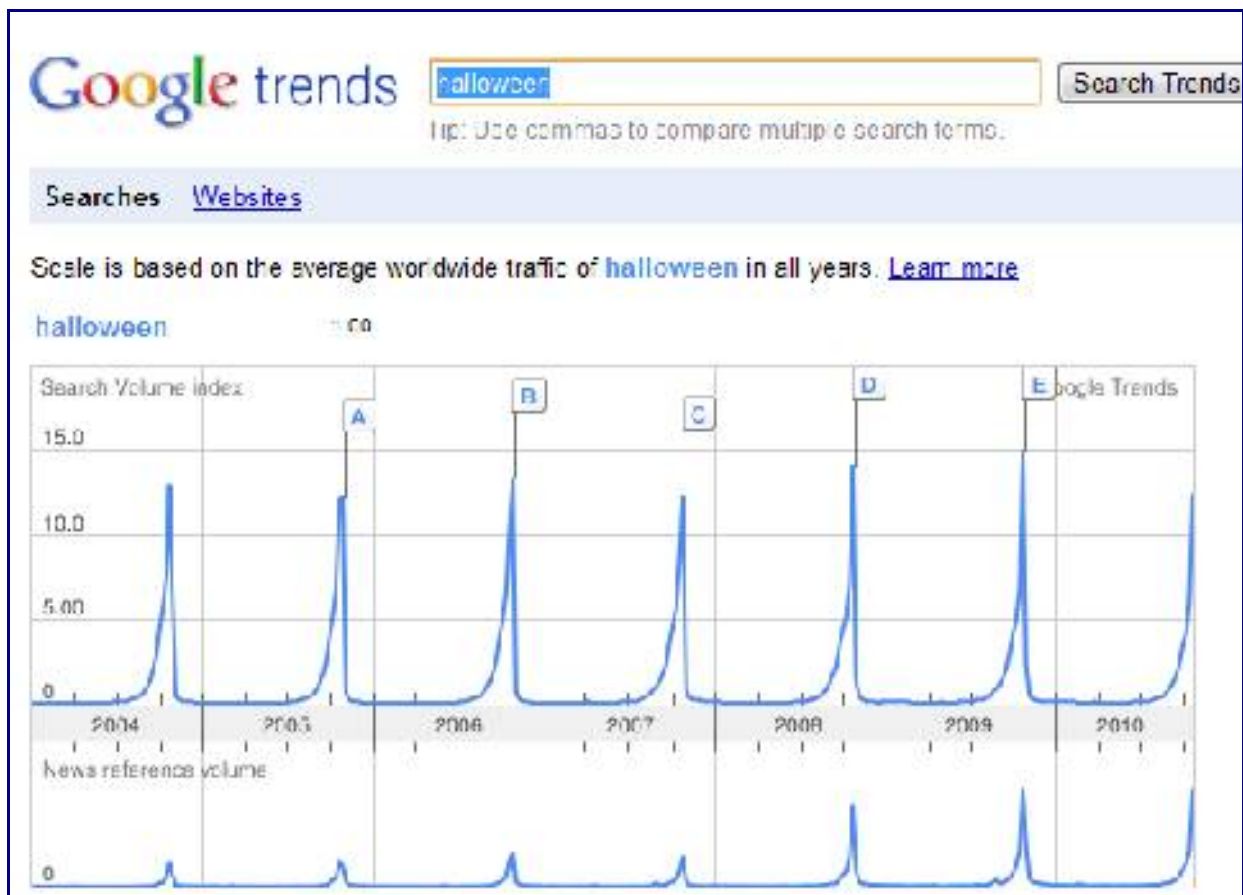
Happy Halloween! Everyone's favorite secular American holiday where kids are in school is invariably the best month of the year for [Bingo Card Creator](#). This year, it ended up being a very happy Halloween indeed. I ended up selling \$6,024.45 worth of software after returns, beating my previous record (set last Halloween) by about 30%. Bingo Card Creator has now sold **over \$100,000 worth of software** (now *that* is spooky) and is on target to hit \$45,000 in sales in 2010.

I did a bit of extra work getting the Halloween promotion to work this year, and also automating/systemizing so that I'll be able to exploit similar opportunities without needing to take time off to do custom coding. Some parts of the strategy worked very, very well. Other parts lagged expectations. And, naturally, I managed to make a critical CSS error and cost myself a few thousand dollars in sales... [again](#).



Timing A Seasonal Promotion

The best time to start preparing for Halloween is in September... preferably, a September *many years ago*. This is because there is a huge, huge surge of Halloween-related queries on Google starting at about October 15th or so, and if you wait you won't have your site ranking in time to ride the tidal wave.



[Halloween costumes], [Halloween parties], [Halloween ideas], etc etc, all follow almost the same distribution. Most importantly for my business, so does [Halloween bingo cards].

A brief digression to understand why this is important: teachers want to play a game on Halloween because it is a children-centered holiday. Halloween is a much more festive occasion in American schools than many other holidays, because kids are not given any time off for it and it is secular. To make a long story short, there is a wide cultural rift in US education about the proper place of religion in public schools, and that makes celebrating e.g. Easter with a rousing game of bingo not exactly a career-enhancing move. Despite Halloween being theoretically based on All Hallows' Eve, in standard American practice it is totally secular in character.

Anyhow, to make a long story short, upwards of a hundred thousand people will go to the Googles and search for [Halloween bingo cards] and a passel of related terms in October. For the last few years I've really wanted to be #1 for that. This year, I was — my mini-site, established in September 2008 or so, finally hit the big time.

Halloween Bingo Cards

Happy Halloween from Bingo Card Creator

Free Halloween Bingo Cards

Halloween is a great time for families to take a little break from the bustle of the daily school year and have some fun. And what more fun than playing some quality-themed bingo with candy as the prize? It makes for an excellent classroom activity as well - you can fit it into the week's lesson plan no matter how busy you are. It is engaging and supports any number of players, and the fast-paced energetic nature helps with kids who might have just a wee bit of a sugar rush.

Our bingo-making abilities have taken us day and night to make a trick or treat for you. Here are printable Halloween Bingo Cards.

There are 6 cards, printed 4 to a sheet to save paper. They look just absolutely like this.

B	I	N	G	O
witch base	ghost	zombie	cat fox	goblin pumpkin
witch ghost	zombie	vampire	ghost	bat
witch ghost	zombie	Free Space	ghost	goblin
candy	monster	bat zombie	ghost	witch
bones	goblin	witch	monster	black cat

Download Free Printable Halloween Bingo Cards

If you can't remember the name you want or download the name right now. (You also might want to grab a call list.)

They are in PDF format. Your computer can probably print them out already, but if not, go get a copy of Acrobat Reader (free). After that all you need to do is print out and play!

Halloween Bingo Activities

- ↑ [Pumpkin R.I.P.](#)
- ↑ [Spooky Scary Bingo Worksheets](#)
- ↑ [Halloween Bingo Game](#)
- ↑ [Halloween Bingo Party](#)

This Week's Most Popular Bingo Games

Most Popular Bingo Cards:
[Halloween Bingo Cards](#)

[Addition 1 To 10 Bingo Cards](#)

[Autism bingo Cards](#)

[Christmas Bingo Cards](#)

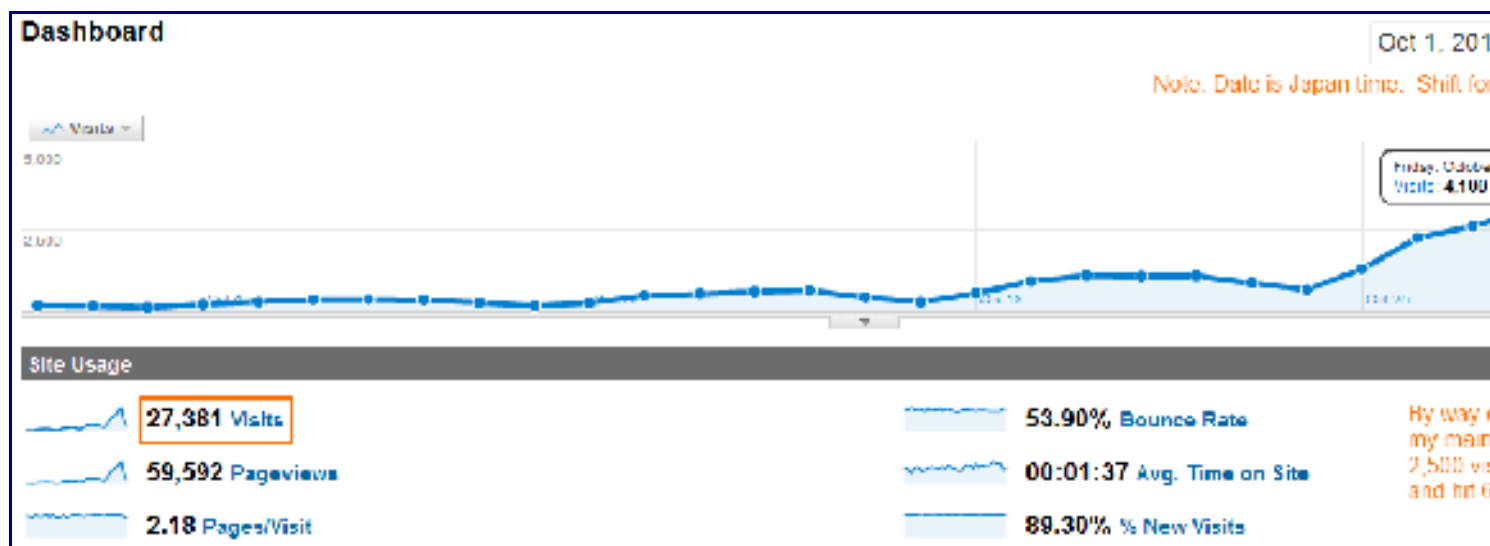
[Multiplication 1 To 10 Bingo Cards](#)

[Bingo 1 and 1 math](#)

You can put this list on your blog.

Use! Go to [Go to the top](#).

Halloween Bingo Cards is what SEOs call a mini-site. It has one mission in life: for two weeks out of the year, it is supposed to be the best, most focused site on the Internet for, well, Halloween bingo cards. Between on-page optimization, a handful of inlinks, the exact match domain bonus (it is a .net, which along with .com and .org receives a huge bonus to rankings if the domain name exact matches the query), and a bit of age, it finally edged out the competition. The extra traffic I got was, well, a **little underwhelming**.



Why was I underwhelmed by 27,000 visits? Well, the site was designed to generate trial signups of Bingo Card Creator, and most of the obvious candidates for action on the site linked people directly to a trial signup form. Conversion rates were unspectacular, in part due to a technical failure describe later and in part because a combination of design issues and just overall low traffic quality meant fairly few people clicked off Halloween Bingo Cards in the first place.

So:

- 27,000 people saw the Halloween mini-site
- 3,500 (**only about 13%**) clicked to the main site (and the signup form)
- 1,300 (**37%**) signed up for the free trial (that is actually extraordinarily good — normally I top out at about 28 ~ 30% for scalable traffic sources)

So of those 1,300 people, how many do you think actually signed up for BCC? Well, since I finally have end-to-end analytics running (through a combination of KissMetrics/Mixpanel and a bit of glue code I wrote myself), I can get away from half-accurate Google Analytics reports and give concrete numbers on this. So I know, with certainty, that that answer is 15 sales **directly attributable** to this page. At about 1.2% conversion that isn't that much to write home about — for the right audiences, BCC gets about 2%+. (It should be noted that Halloween traffic is almost always going to be the wrong audience, because most *searchers* are parents but most of my *customers* are teachers: parents have their needs 100% met by the free trial.)

So What Worked Better Then?

Email. Holy cow, email. This has been a blindspot of mine for years since I used to be an anti-spam researcher, do not really send or read mail that regularly outside of doing customer support, and hate newsletters with a passion. **Big mistake:** my customers empirically do not feel the same way.

Halfway through last year I signed up for [MailChimp](#) with the intention of doing great things with it, but I struggled to get it to work out right. I would lose interest for several months at a time and not send email to the list, which causes people to forget who you are and then get very peevish when they receive mail from you. About the only thing that worked very well for me was my auto-responder, which automatically mails people 1 and 6 days after they sign up for it with hints and tricks, to basically remind them that I am still here. (A huge percentage of BCC users never sign in after their first day, and anything I can do to remind them of my existence is worth serious money.)

Sadly, I got an automated notice from MailChimp several months ago about excessive unsubscribes from my autoresponder — about 1% (they're **serious** about list quality at MailChimp), and paused it while editing to make it obvious who I was and why they were getting email from me. (The phrase “because you went out of your way, *twice*, to ask for it” may have been in an initial draft.) And then I left the autoresponder paused for several months. *sigh* So I wrote a new alert for my dashboard about that, since I'm serious about making mistakes only once.

Anyway: [Rob Walling](#) had an amazing [presentation](#) at the Business of Software conference about using email to sell more software, and I was so inspired I decided to get serious about it this year.

- I used MailChimp filters to go down from the 8,000 or so people on my list to only the 800 signing up since June, figuring they would still remember me. (In hindsight, I should have cut further, and I will make a point about absolutely, positively keeping the list “fresh” from here on out.)
- I tweaked my from address from “Bingo Card Creator” to be “Patrick McKenzie (Bingo Card Creator)”, at Rob's suggestion.
- I mulled over reasons why teachers might open the mail, and settled on “A Halloween Activity (And Discount) From Your Friends at Bingo Card Creator”

The time-limited discount was repeatedly stressed in Rob's presentation and in his [book about selling software](#). (He gave me a free copy. You should buy it just for the email chapter.) So I spent a few hours tweaking my shopping cart so it could use discounts, and then polished until the experience of receiving them was totally transparent to the user — they never have to fumble for a code, all they have to do is click a link in their email and they're in.

BINGO Bingo Card Creator

Make Bingo Cards On Your Computer!

Your Home Settings Buy Bingo Card Creator Support Contact Us Logout

Word Lists

Open a word list to start printing bingo cards.

Create new Word list

Use Pre-made Word lists

You have no saved word lists yet. [Create a new word list](#) or [use a pre-made word list](#) to start making your bingo cards.

Happy Halloween! Buy Bingo Card Creator before November 1st and we'll take \$5 off the price! Hurry, you only have 0 days left!

Looking to create Halloween bingo cards? [Click here to get started.](#)

Free Trial of Bingo Card Creator

Here's what you need to do to print bingo cards:

1. [Create a new word list](#) or [use one of ours.](#)
2. [Customize your bingo cards.](#)
3. [Download the cards and print them.](#)

That's it!

This discount was extraordinarily effective at motivating sales, relative to many things I have tried.

- 775 mails sent.
- 170 (23%) opened
- Of those, 6 people bought.

At a little less than 1% conversion from an *email* to a sale, I'm absolutely giddy about the future potential for email marketing. I got perhaps excessively giddy and sent another email two weeks later to people who didn't open the first one. At least, that was my intention: I actually hit the button for sending it to people who didn't open last year's Halloween newsletter (i.e. essentially everyone who had just been mailed two weeks prior). *Doh.*

One person did not appreciate getting two newsletters in one month after not getting them for several months. I got a handful of spam complaints for that (four from the first send, one from the second).

Since MailChimp is hypersensitive to spam complaints and I likewise care about my reputation, I'm going to be more careful in the future. I anticipate the best thing to do to keep them down is to just email people early and often to keep myself in their mind, which is almost the opposite of my natural inclination.

Anyhow, the two campaigns together:

- 1,600 emails (total cost: \$32 of MailChimp credits)
- 60 reactivated accounts
- 15 sales (~ \$370 in revenue)

Very little I have done has **scalable 1,000% ROIs** on marketing spends. You can bet I will be doing more of this in the future. (Granted, this doesn't count approximately 3 hours of effort upgrading my shopping cart and site to do discounts, but I can amortize that over every time I use them from here on out. The newsletter was the 2009 Halloween newsletter with about 5 minutes of editing.)

Speaking Of Discounts

So after having huge results early in the month with the emailed discount offer, I thought “Hey, why not extend that to everybody? I'm getting crazy 10% conversion rates on reactivated accounts — if I got 10% conversion rates on new accounts I would be making hats out of money.” Out of an abundance of caution, I A/B tested that for the last two weeks. Surprisingly, **discounts failed to make a dent in sales numbers** in the general (non-email) population. Roughly 2.08% of people seeing the discount converted, and 1.87% converted without seeing the discount. That is despite prominent notices on the first screen after you log in, plus the pricing page, and the exploding nature of the discount (act now or you won't get it). Not only is that difference below the floor of statistical significance, when you factor in the fact that I make \$5 less for every discounted copy, the full-price version did better for me.

Things to try next time:

- Only give the discount to people who I'm reactivating — their accounts are writeoffs if they don't sign back in, after all.
- Pick the cutoff date for the discount better. I gave people to November 1st, because Halloween is the 31st and I wanted to be generous. However, that diminishes the perceived urgency of the sale — on Thursday night, they still have “3 days left!” **That was stupid of me** because I know, from doing this for several years, that my window of opportunity for Halloween sales closes as soon as she goes to sleep on Thursday. I should have had the Halloween sale go until Thursday night, which makes it a pretty cruddy “Halloween” sale but would almost certainly have goosed sales.

That said, that was a pretty good day for me — Wednesday before Halloween was my best day ever, right until Thursday smashed that record. Last Halloween I was beginning to pull weeks upon weeks of crunch, and I think I probably made less in a 70 hour workweek than I did in those two days... while sleeping. I love self-employment.

Google Enjoyed Halloween, Too

While I was #1 on Google for [Halloween bingo cards], [Halloween bingo] and a thousand words on the long tail also receive significant traffic, and many of them are sewn up by about.com, eHow, etc etc. Luckily, most of the content mills run AdSense ads, and the guy at the front of the auction screaming “Pick me pick me!” was yours truly. I spent \$1,764.84 on AdWords ads in October. Sadly, much of it got wasted.

Why? Well, I made one extraordinarily poor decision and followed it up with a mistake: when FireSheep (the sniff-your-credentials Firefox plugin) was announced, I immediately bumped SSL support from “something I'd eventually like Bingo Card Creator to have” to “must be done today”. If I had had any sense, I would have let that wait until November. Enabling SSL took about a day, but I missed some edge cases which came back to cut me: in particular, I had the registration form throwing SSL errors for about 6 hours (painful, particularly since it was during an email delivery window) and, even more painfully, I had the AdWords landing page throwing errors for 48 hours (not fun when

you're spending \$150 *a day* driving people to it).

But the worst thing was keeping the freaking AdWords conversion code on HTTP, which caused some browsers to not load it from an SSL page, and cut my conversion rate in half. Since I use Conversion Optimizer, this caused Google's algorithms to think "Hmm, Patrick must really not want this flood of traffic we're sending him since it apparently converts like crap. Oh well. I guess we'll just send it somewhere else instead." Between money that I essentially set fire to, an AdWords campaign thrown into disarray (I still haven't recovered my previous conversion rates, since the bots are in a bit of confusion and haven't replaced me on my best performing sites yet), and missed opportunities, that mistake probably cost me a few thousand dollars. Ouch.

Still, I generally try to see things as half-full rather than half-empty, and my jack o'lantern is overflowing at the moment: previous sales records smashed, a new scalable marketing tactic which actually works, infrastructure ready to go for Thanksgiving, Christmas, and Valentine's Day (and a few hundred more email addresses to send to), and the SSL issues now *mostly* under control. (I still am not transitioned to 100% SSL, which ironically means that the effective increase in security was minimal. *sigh*)

Mini-update on Appointment Reminder

I was very busy with ongoing improvements to Bingo Card Creator, consulting, travel, and non-business life for the first six months of being self-employed. Since midway through October I've refocused to get Appointment Reminder out the door (really — most of the Halloween campaign was taking advantage of things which already existed), and since previous experience has taught me that artificial deadlines really work for me, I have joined up with a bunch of Hacker Newsies to make November the Get Your Startup Accomplished month. After some fumbling around with jQuery and other infrastructural issues, I feel like I really hit my stride today, and if I keep it up I should launch right around the end of the month.

If you'd be interested in hearing about this on a regular basis, leave me a comment — if folks care, I will post regular updates to the blog.

Posted in [marketing](#) | [32 Responses](#)

[How To Use SSL To Secure Your Rails App Against FireSheep And Other Evils](#)

By [Patrick](#) on

The post on Hacker News today about [FireSheep](#), a Firefox addon which lets you trivially compromise the web application cookies/sessions of anyone on the same wireless network, gave me the much-needed impetus to upgrade my business to SSL security. For the last several years, I haven't encrypted traffic between the server and the user. My theory was that my customer's didn't store anything security critical in their elementary school bingo card games, but the increasing amount of information available to the admin (me) plus a customer support story from this morning convinced me that compromise would be a Very Bad Thing.

Implementing SSL in Rails was very painful and resulted in my site being down or unusable for a large portion of my customers for much of the day. (If I were doing it again, I would have paid the extra few bucks to set up a staging environment with its own certificate and verified everything worked on that prior to trying to fight my way through the process.) Luckily, since I am time-shifted from them

by over 12 hours, few noticed. In the interests of saving you and your customers some difficulty, I thought I would write up what I learned.

What You Need Before You Get Started

1. A SSL certificate from a certificate authority which is trusted by the major browsers. [GoDaddy](#) sells them for \$12.99 and they are perfectly adequate.
2. [SslRequirement](#) and [AssetHostingWithMinimalSsl](#), both plugins by DHH.
3. Rails to be fronted by Nginx. The explanation for Apache is similar but the magic server configuration is different. (If you use Nginx+Passenger, can skip the Mongrel-specific bit below.)

Nginx configuration

Nginx manages configurations on a per-server basis, and cannot have a single server declaration listen to both HTTP and HTTPS requests. We're going to get around having to copy/paste our entire configuration (and maintain it in two places) by DRYing it into a single external snippet, then including it twice.

Right now, your Nginx config looks something like this:

[view sourceprint?](#)

```
1.server {
  2.     listen          80;
3.
  4.     //You have a lot of stuff here.
5.}
```

Cut everything out of the server declaration (yes, everything) and externalize it into a separate file. It should now look like:

[view sourceprint?](#)

```
1.server {
  2.     listen          80;
  3.     // This path is relative to the conf directory
  4.     include apps/EverythingJustCut.conf;
5.}
```

You can now create a separate declaration for your SSL server without causing much overlap:

[view sourceprint?](#)

```
01. server {
02.     listen 443;
03.     ssl on;
04.
05.     #GoDaddy's instructions will walk you through setting these up
06.     ssl_certificate
    /usr/local/nginx/conf/ssl/your_certificate_combined.crt;
07.     ssl_certificate_key
    /usr/local/nginx/conf/ssl/your_key_pair.key;
08.
09.     include apps/EverythingJustCut.conf;
```

10.}

I remember setting up the SSL certificate to be more of a nuisance than a difficulty, but if not, you can find [very detailed instructions](#) online.

Now, **if you are using Mongrel**, you need to do one more thing: withing EverythingJustCut.conf, you've got to find the place where Nginx is proxying to Mongrel and have Nginx set the X_FORWARDED_PROTO to https for HTTPS requests. This is the one and only signal that Rails, running on Mongrel, is going to get that a particular request is for SSL or not. This is trivial to do if you know you have to do it: just find all your existing proxy_set_header statements and add this after them:

[view sourceprint?](#)

```
1.proxy_set_header X_FORWARDED_PROTO $scheme; // http or https,
evaluated per request by Nginx
```

Very Carefully Scrub Your Website For Absolute URLs

Absolute URLs containing the scheme (i.e. anything starting with http://) are dangerous to your application, because if your page transmitted over HTTPS references a “certain type of resource” on HTTP, your browser may display a Scary Error Message to your users. Unhappily, the browsers get to pick what constitutes a security-critical resource.

The general rules for maintaining SSL security on HTTPS pages are:

- **Javascript files**: must always be loaded from HTTPS
- **Image files**: must always be loaded from HTTPS, **except** for Firefox and Safari
- **CSS files**: must always be loaded from HTTPS, **except** for Safari
- **other files**: may or may not be loaded from HTTPS

Do you think you're going to remember that? Yeah, me neither. Hence, you don't want any hardcoded http:// anywhere. Let Rails take care of it for you with AssetHostingWithMinimalSsl: all you have to do is be consistent about always loading e.g. images through the image_tag or image_path helper, CSS and Javascript through their associated helpers, etc, and you will always get the proper behavior. The tough part is that you're going to have to take IE and drive through every page on your site verifying that it does not accidentally include a resource transmitted in the clear.

The configuration for AssetHostingWithMinimalSsl goes in your config/environments/production.rb file and is trivial:

[view sourceprint?](#)

```
1.config.action_controller.asset_host =
AssetHostingWithMinimumSsl.new(
  2. "http://images%d.example.com", #In this example, I have
  images1..images4 .example.com configured in DNS
  3. "https://www.example.com" #Your SSL-secured domain
4.)
```

Note that it is **very easy** to miss a http:// URL hidden in a CSS file, Javascript file, or analytics-package JS include somewhere. If you do that, even in an unused CSS selector, you will cause the browser to throw Big Scary Errors. Test that you have gotten everything **very thoroughly** prior to proceeding.

Require SSL for Any Security Sensitive Actions

Why are we requiring SSL? To prevent against an attack where the bad guy can sniff the cookie out of

there air, thereby appropriating it for himself and logging in as the logged-in user (either by compromising a session ID or, in Rails, compromising the user_id you probably stored in the tamper-proof CookieStore). So what do we have to SSL? Everything Rails sends or accepts a session cookie with. What is that? **Everything** that an actual browser can access. (If you are, like me, in the unenviable situation where some URLs are going to be hit by user agents which cannot support either HTTPS or cookies, don't forget that requiring SSL for all actions won't help you.)

The SslRequirement plugin makes it easier to do this sitewide than it otherwise would be:

[view sourceprint?](#)

```
01.#in application_controller.rb
02. unless RAILS_ENV == "production"
03.   def self.ssl_required(*splat)
04.     false
05.   end
06.
07.   def self.ssl_allowed(*splat)
08.     false
09.   end
10. else
11.   include SslRequirement
12. end
```

This sets it so that SSL is required when we say it is in production only, and in other environments the statements which set up SSL requirements are merely silently ignored. Those functions are:

- **ssl_required**: takes a list of :symbols representing action names to require SSL for. Should be nearly all of your actions. If someone tries to access one of these actions over HTTP, they will be redirected to HTTPS (+).
- **ssl_allowed**: takes a list of :symbols representing action names to allow SSL for. If they aren't required and aren't allowed, then they'll be redirected to HTTP if they try getting to this action over HTTPS.

You set them in each controller, on a per controller basis. There does not appear to be a handy mass assignment option like :all for this method, unlike most of the before_filter and similar things you find in Rails.

Note there is a subtlety here: if you let Rails share a session cookie over both HTTP and HTTPS, then if it is ever used over HTTP, byebye cookie security. This means you can either be very, very careful that you never let anyone access Rails actions over HTTP (and you pray a malicious attacker never tricks your users into clicking to a **valid link to your website**), or you ban your session cookie from being sent over HTTP at all:

[view sourceprint?](#)

```
1.#production.rb
2.config.action_controller.session = {
3.  :key => 'name_of_session_goes_here',
4.  :secret => 'you need to fill in a fairly long secret
here and obviously do not copy paste this one',
5.  :expire_after => 14 * 24 * 3600, #I keep folks logged in for
two weeks
6.  :secure => true #The session will now not be sent or received
on HTTP requests.
```

7. }

This option will **probably break your site**, possibly subtly, the first time you switch it on. Test thoroughly. I haven't got it 100% working for myself yet.

Host downloadable files on SSL? You just broke IE.

After several hours of frustration, failing my way forward, and finally getting things working on Chrome/Firefox, I received a bug report from an IE using customer who couldn't download her bingo cards. Some deep Googling revealed that IE, for architectural reasons known only to the IE team, does not play well with downloadable files over SSL unless you set some very specific headers:

[view sourceprint?](#)

```
1. response.headers["Pragma"] = " "
2. response.headers["Cache-Control"] = " "
3. response.headers["Content-Type"] = "application/pdf" #Put your
   favorite MIME type here
4. response.headers["Content-Disposition"] = "attachment;
   filename=#{filename}" #
5. response.headers["X-Accel-Redirect"] = "/path/to/file.pdf"
6. render :nothing => true
```

Note I am using X-Accel-Redirect to have the file served directly through Nginx, rather than through Mongrel, as described [here](#).

Conclusion

I hope that saved you and your customers some pain and insecurity. If you haven't done this yet, and I think there are many Rails apps as open to exploitation as I was until this afternoon, I suggest you download FireSheep and see how quickly any idiot with wireless can compromise your admin session. Then, **fix this** as soon as possible.

Posted in [Rails](#), [Ruby](#) | [24 Responses](#)

[Lessons Learned At Business of Software 2010](#)

By [Patrick](#) on

Author's note: This post is gigantic — over 10,000 words. I don't recommend reading this in one sitting — bookmark it and come back later.

[Bookmark this page on Delicious, saved by 173 other people](#)

Tags:

- [blog](#)
- [business](#)
- [microisv](#)
- [entrepreneurship](#)
- [startup](#)
- [software](#)
- [marketing](#)
- [programming](#)

- [startups](#)
- [seo](#)

Early this October, I was privileged to attend the [Business of Software conference](#) as both attendee and a speaker (see disclaimer [waaaaay at the bottom](#) of this post). It was far and away the best conference I've ever been to. You should go to it next year — I know I will, if I have to swim from Japan to Boston to do so. (If you would like to but don't think it is possible, see [waaaaay at the bottom](#) of this post.)

The featured speakers were, virtually without exception, remarkable. The 7.5 minute Lightning Talks (one of which I presented — more about that after they post the video, you will be amused or your money back) were often entertaining and at least a few of them were packed with better detail than 15 Powerpoint slides have any right to be.

The phenomenal content of the conference, however, is just an excuse to get a few hundred ferociously smart people in the same hotel for three days. I got to finally meet in person a few of the folks who either personally assisted me in my early days as a businessman or have been tremendously influential in my thinking — a short list of shoutouts includes Ian Landsman, Dave Collins, Joel Spolsky, Eric Sink, Eric Ries, Peldi from Balsamiq, and I could put another dozen names here without breaking a sweat. One speaker at one point asked all attendees who had founded a software business to stand at one point, and by my eye probably 60% of the room stood up. During breakout sessions, I heard from everyone from folks doing small software businesses like myself to a company which sells monitoring software for nuclear power plants.

The spirit of generosity at the conference was off-the-charts amazing. However, many conversations were either by explicit agreement or common courtesy off the record — not everybody publishes sales stats, and that is their right — so I'm going to err on the side of caution and only repeat things I learned from the public sessions. That said: *beg, borrow, or steal tickets to it next year.* I can't repeat that enough. Although the conference swung to business topics far more often than technical topics, almost everyone I talked to both inspired *and* holding a notebook full of actionable advice to try.

My notes are incomplete and my memory is fallible, but here were some of the takeaways I got from the public presentations. These are more a combination of my notes and thoughts than transcriptions of what was said. If you want those, wait for the videos to be released.

Seth Godin

Seth posted a [riff on his remarks](#) on his blog. The general tone and content was not surprising if you've followed him for a while. (If you don't, do so — aside from being a great communicator, he's one of the better big-picture thinkers on marketing that I know of.)

Specifically, he says that the software business is undergoing radical changes because technical competence is no longer a scarce commodity: with open source tools, an increasing supply of trained programmers, and the explosion of cost-lowering innovations for creating and marketing software on the Internet, things which were previously the purview of a technical elite are now within what talented teenagers can cook up from their kitchen table. The marketing side of the equation is also changing, since increased crowding means that the traditional model of paying salesmen to push software has devolved into using humans as inefficient spam bots. (See his whole spiel about Permission Marketing here.)

A brief guide to evaluating opportunities:

- Who can I reach with my marketing message?
- Will they talk to people on my behalf?
- Can I earn and maintain permission to continue the conversation with them?
- Will they pay for my solution to their problems?

Perceptive readers will note that technical difficulty appears nowhere above.

Best four words of talk: “Ideas that spread, win.” Software is now about creating a tribe or otherwise achieving leadership of one. Take AutoCAD, for example: it isn’t a product, it is a *movement*: we architects who use AutoCAD are here to change the industry away from using pencil&paper drafting.

You are either in the movement or opposed to it, because it is an existential threat to your business, but you certainly aren’t neutral on the question. Boring software creates no movements and has to be sold the old fashioned (expensive, ineffective) way.

Another example of movements is the trend towards measuring things, which was a recurring theme during the conference. Toyota succeeded in changing the face of manufacturing because they got religion about statistics, and a similar sea change is finally happening in software. This meta-trend is creating a lot of extraordinarily valuable companies directly and also making the rest of us leaner and meaner. (Bingo Card Creator has a more sophisticated approach to metrics than many Fortune 500s. That’s a shame, but also a *tremendous* opportunity for smart, savvy software firms.)

I was quite amused by a throw-away example of the difference between capitalists and socialists inspired by pin-making machines: prior to the industrial revolution, it took a skilled artisan a day to hone 5 pins (small bits of metal which one uses to, e.g., attach fabric to each other) by hand. A pin-making machine lets any 5 people who can be trusted to learn a simple, repetitive set of instructions make 10,000 pins a day. Socialists hearing this thought “Oh no, that is going to totally destroy the livelihood of pinmakers.” Capitalists hearing this thought “Memo to self: be the guy who owns the pin-making machine.” As the cost of software approaches zero, you want to be the guy who owns the machine.

Relatedly, with regards to your future employment prospects: draw a Venn diagram of job skills you have and skills which can be automated (or, by extension, anything which can be adequately explained by any amount of writing on paper, because anything that can be written down can get shipped to India). Anything in the intersection represents career suicide.

Seth claims that no “competent” people work at his current software company, Squidoo, because “if we only needed *competency*, we can get that from [outsourcing]“. You hire exceptional people and let them execute on things which require exceptional talent, and write down things which only require competency. Low-wage grunts can do the competent stuff. ”Your job is a platform, not a list of tasks.”

Some personal thoughts: Seth was bearing on charging for software because of competition from OSS competitors, preferring instead to lead movements where joining was free (to maximize the size of the movement). Happily for those of us who sell software, almost all OSS projects are [incompetent at marketing](#), which matters more than technical execution. Seth gave an example of trying to sell water: what are you going to do, make water wetter? That is a wonderful example, since bottled water didn’t exist as a category 50 years ago and sales are going up every year, over a century after America solved the problem of giving everybody what amounts to *free, infinite, drinkable water*. It amused me greatly to hear that bottled water was going away while attending a conference at a luxury hotel which sold \$4 bottles of bottled water placed in the restroom *literally next to the tap* of ever-flowing unmetered municipal goodness.

David Russo

David gave a presentation about hiring at software companies, in particular about hiring to create and sustain a corporate culture as the company approaches scalability. I regret I was insufficiently attentive during it, as this topic is several years ahead of need for me.

Briefly, there are five models for corporate culture in the literature:

- Bureaucracy (the traditional BigCorp model)
- Autocracy (Apple-style, where the company is an extension of the Will of Jobs)
- Engineering (Google, where the engineers run the asylum)
- Star culture (a large consulting firm or, perhaps, 37Signals, where individual contributors are expected to be stellar and you can put up with rough edges because, hey, rockstars)
- Commitment — a consensus culture with loyalty receiving strong priority (and, as you can probably guess by position on the list, the clear favored choice of the presentation)

There was an example of Westinghouse having a non-engineering CEO who proved to be disastrous for the company. Afterwards, the board resolved never to hire a non-engineer again, because the culture of Westinghouse was ferociously engineering-dominated and when engineers talk to each other they break out the secret engineer decoder ring. The CEO, who couldn't speak the language or achieve respect among the engineers, was sidelined and failed at achieving his objectives.

Culture generally grows organically out of a business rather than being imposed from the top-down.

Changing a firm's culture midstream is very painful, and Harvard studies suggest it has quantifiable negative impacts: 50% reduction in odds of an IPO, tripled likelihood of business failure, and 3% less growth *monthly*. (The engineer in me winces at the compounding there.)

David's advice was to invest in employees: hire brilliant people, and let them be brilliant. The outcomes they achieve are more important than the work itself.

Dharmesh Shah

Dharmesh is CEO (most recently) of [Hubspot](#), a marketing tool for small businesses. They have raised quite a bit of money and, more importantly, have a lot of paying customers, consistent growth, and a revenue curve which virtually defines "up and to the right". You should be reading [his blog](#) already.

He presented a few techniques from Hubspot which were "interesting and not obvious". For example, successful SaaS companies which IPOed had a median capital raise of \$42 million dollars. It doesn't match with our expectation that you can make a SaaS startup in your garage, partially because SaaS companies essentially offer their customers upfront financing (by spending their cost of customer acquisition in a front-loaded fashion and then recouping it over the 4-5 year lifespan of that relationship).

A key metric to keep track of: customer churn rate, because a 50% decrease in it doubles customer average lifetime value (LTV). One of Dharmesh's periodic hobbyhorses is that if you know LTV and know the cost to acquire a customer (CAC), then if CAC is less than LTV by a fair bit, you take as much investment as humanly possible, spin straw into gold, and walk away with money hats. (This is far and away the best pitch for a software company taking VC money that I've ever heard. See also the [video of him speaking last year](#), if I remember right he covers it there.)

There are a few ways to measure churn: as percent of customers, as percent of revenue (better accounts churning faster than entry-level accounts: uh oh!), and "discretionary churn", which is churn among accounts who are actually able to make the decision to cancel and are not, e.g., locked into a one-year

contract with you at the moment. However, overoptimizing on churn rates is myopic, because **absence of churn does not indicate the presence of delight**, and customer delight is a key to long-term success at marketing the product.

One particular way Hubspot maximizes for customer delight is by creation and use of a Customer Happiness Index (CHI), which is a backtested predictive measure of someone's likeliness of churning next month. CHI is dominated by three factors and adjusted as they get more data:

1. Frequency of use (more use = less churn)
2. Breadth of use (customers who use more features churn less)
3. Use of "sticky features" (there exist some features which are so screamingly better than alternatives that a customer who uses and gets value from them is very reluctant to stop using the service)

If I can call out #3 for a moment: many companies I have worked with have "sticky features", although not all of them know it. (Some day when I don't have another few thousand words to write I'll tell you about what they are for Bingo Card Creator and how I know.)

After you have CHI available in a transparent fashion throughout your organization, you can exploit it in a variety of ways:

- Tell your sales representatives to call customers with low CHI proactively and ask them what you can do to make them happy. This simple technique saves 1/3 of canceled accounts, and is virtually free.
- Judge sales reps not just on sales but on CHI of new accounts. (This counters the incentive to armtwist to sell to marginal customers.)
- Judge new features / policies / etc based on their impact to CHI. Concerned about whether charging for consulting for setup is a good play for the business or not? A/B test doing it versus not doing it, and optimize for CHI. (I get the feeling that they might not explicitly be doing A/B testing... but they should be. Sorry, hobbyhorse of mine.)

All meetings at Hubspot include a teddy bear named Molly, who is a stand-in for the customer (they have two customer archetypes: Margaret the marketing director and Olly the owner, collectively, Molly). When they have contentious debates internally, they are frequently decided by someone saying "Hey, what would Molly want us to do here?" Brilliant hack, practically free.

Hubspot sells services, primarily consulting for setting up new clients. This represents a fraction of their revenue — only 7% or so — and decreases their gross margins, because consulting services are intrinsically lower margin than selling software products. But "services are low margin... except when they're not": since getting people set up for success increases their CHI and wildly decreases churn rate, selling someone a few hundred bucks worth of engineer time at low margins can lead to thousands of dollars of marginal increase in LTV delivered by high margin renewals of the core product (which has 90+% margins). **Star this advice, gentlemen, because it has obvious implications for B2B SAAS startups.**

So if you know customers and the company benefit hugely by increasing consumption of consulting services, why charge for them? Answer: perceived value increases consumption, and perceived value is driven by cost. When you give people 5 hours of free telephone support with an engineer, they — being successful businesspeople — find difficulty clearing 5 hours free in their schedule, fail to show up for the call, get distracted, etc. When you charge them \$500 for the call, **they damn well show up.** (Patrick notes: why *hello*, [recent discussion](#) about [Chargify free customers and support costs](#).)

In terms of explaining the value proposition of your software to customers, cribbing from Kathy Sierra: don't make _____ software, make _____ superstars. Make your customers awesome at their jobs /

tasks, and they will pay any price and/or follow you to the gates of Hell.

Hubspot is big about being transparent to a fault within the company, particularly by using a company-wide Wiki which includes financial details, plans, and pretty much everything except salary information. (Apparently, that being public is viewed as possibly toxic to morale without having obvious upsides. Having financial details open to employee audit has obvious upsides: employees can tell you when you've let the entrepreneur reality distortion field go wild in the board meeting.)

Best sentence about branding I've heard recently: "Brand is what people say about you after you have left the room."

Eric Ries

I will refrain from giving an in-depth summary of this talk because if you have read his blog you pretty much know the gist of it before. If you haven't read his blog, [start doing so](#). With the obligatory disclaimer about letting any methodology take over your higher brain processes, the Lean Startup movement (and some of the specific, implementable tactics it suggests) are so effective I sometimes worry about whether established competitors will try to illegalize them. The best effect of this presentation was taking the idea away from the Silicon Valley echo chamber (where "pivot" has become something of a buzzword) and bringing it to an audience of people who largely sell things to people for money, but who still could benefit immensely from some of the Lean Startup lessons.

If you want an hour-long video introduction by Eric to the Lean Startup, you can see parts [1](#) and [2](#) of a previous explanation on the Internet.

Some specific lessons I haven't heard before in a couple years of lurking on his blog or I thought were worth highlighting:

- The key takeaway: Lean Startup matters because the traditional model of development **wastes people's lives**. They pour out years of hard-charging startup workweeks making software which doesn't actually improve the lives of any customer because it solves a problem no one has. This waste of potential is criminal and can be fixed.
- Machine shops and software firms are self-regulating ecosystems if you do them right. For example, if you use continuous deployment and five whys, when you start doing development too fast, you'll get the brakes applied by having to make investments in quality control repeatedly because you're making too many mistakes. This lets you find the optimal pace of learning without having to fiat an inaccurate number ("You will write 10,000 lines of code a week!" — which as we all know means nothing, and definitely nothing good) from on high.
- The Agile methodology is a wonderful solution to the peculiar problems of internal IT departments at big companies, where the problem is extraordinarily well-known ("We need our existing payroll system, except on a new architecture where extending it doesn't require sacrificing goats") and solutions are unknown ("What is the Rails code to do this?"). It has less to offer startups, where **both** the problem and solution are unknown. ("Do people even need the thing we're fumbling forward on building right now?")
- Since the only people you can plausibly get to use a new product are early adopters, any additional quantum of work done to get the product ready for mainstream users is a form of waste. (This is a bold statement with interesting implications. I'm not sure I agree with it, but it sounds potentially very powerful.)

Scott Farquhar

The president of [Atlassian](#) had some thoughts gained from bootstrapping the company from 2 people to a \$60 million capital raise and beyond. Regrettably for my notes, this talk was structured as a List of N Things, which ensured that I wrote down N things and grabbed few supporting details.

1. Start with 2 founders. For example, when their password database was compromised while Scott was off on his honeymoon in the wilds of Africa, knowing that his cofounder had the situation under control alleviated a portion of the anxiety, and after an emergency flight back to Australia Scott was able to let the cofounder get some sleep.
2. You need a business model.
3. Dogfood your products.
4. Measure everything.
5. Always be marketing. Atlassian excels at this — for example, they had a charity give-away where they gave licenses of JIRA (a bug tracker) away for a song to small companies (\$5 for 5 licenses or something to that effect), donated the profits to charity, and got massive good press from the deal while also increasing sales in the market segments which drive most of their actual revenue. They also had very appealing marketing of another few initiatives: calling something the Stimulus Package, for example, was an excellent PR hook in the financial crisis environment. I wish I had better notes here. Memo to speakers: please do not autocommoditize your best points by numbering them.
6. Your first idea will fail. Atlassian was originally going to market plugins for an obscure product from a company in Europe. It turns out the bug tracker they wrote to make that product was more valuable than the product itself — shades of Fog Creek and CityDesk here. (We heard this theme several times during the conference: Peldi later said that Mockups was originally intended strictly as a plugin for another software, and the standalone version which it grew into now dominates Balsamiq's revenues.)
7. Long term thinking is good.
8. Know when to switch gears.
9. Build somewhere you want to work. Many precious anecdotes here about, e.g., organizing a city-wide scavenger hunt for employees.
10. Experiences are more effective at motivating employees than and customers than money. This comports with both social science research I've read and my own experience. (I have recent experience of this at a company I consulted for, but since it is identifying, I'll wait until they announce what we did together to go into it. Still: it is amazing how effective turning money into an experience is at motivating people than awarding them the money directly, even though this is crazily irrational. Japanese companies understand this better than anyone: "why pay people so that they can buy objects suggesting social status when you can just award social status directly", which is the Rosetta Stone for so much of corporate life here it is scary.)

Jason Cohen

I was flagging by this point in the conference and a little terrified about my upcoming speech, so my notes aren't that great. Jason had a lot of amusing anecdotes from taking Smart Bear code review software from a one-man operation in a small room in Texas through to a sale to another company. (Despite being told, repeatedly, that "Smart Bear" was unprofessional and the name would scare off big Fortune 500s, the company which bought him out later reorganized their portfolio of products under that brand name.)

One amusing story was about Sales Guru Frank. He was a silver-haired slick talker who, for a mere

50% of the equity of the company, promised to sell Smart Bear software to the sorts of huge corporate clients which would bring it from a six figure business to a smashing success. Jason had an absolutely priceless anecdote about how us engineers perceive the sales process: “You know, it’s when you’re in the boardroom drinking some expensive brand of alcohol and you start the conversation with ‘Hey, did you see the game?’ and *everybody understands what game you meant* and then, suddenly, sales happen.” It turns out that sales is a little more complicated than this (for details, see the Paul Kenny presentation) and, thankfully, Jason avoided getting suckered by Frank.

Similar fun stories abounded but I was too terrified of my upcoming presentation to write them down (d’oh).

The single best takeaway was how to evaluate advice you receive from noted software bloggers.

There are [two competing motivations](#) for people who start software companies: wanting to maximize one’s financial outcome as quickly as possible (you want to be **Rich**) and wanting to sculpt the perfect personal niche in which you will be respected and enjoy your day-to-day work (you want to be **King**).

There are two fairly well-known markets for software: **B2B** (business to business) and **B2C** (business to customer).

This gives us the traditional four quadrant graph, which Jason decorated with some examples:

	RICH	KING
B2B	Smart Bear (presenter) Slack Overflow (maybe)	37Signals Fog Creek
B2C	Mint Trendy Valley Startup #03	Balsamiq Bingo Card Creator

The take-away: only listen to advice given from people in the same quadrant as yourself, and you’ll save yourself a lot of pain and cognitive dissonance from getting conflicting advice. (This is overstating the conclusion a wee bit I think, and people change on both axes over time. That said, I’m definitely onboard with “all advice is a product of the circumstances faced by the person giving it”, but I think you can pull lessons from 37Signals in a B2C business or from Joel Spolsky regardless of whether you want to craft the ideal place to work or go big with a VC company or both.)

Sidenote: my business got mentioned from the stage three times, and I’m seriously honored and humbled about being put in that company.

Paul Kenny

Paul talked about the sales function — not just the sales job title — at software companies. See his [talk from last year](#) for details about that.

Why do so many salespeople suck? Well, the typical career path requires:

- One year from hiring to get up and running with the particular company/market segment/product.

- One year to achieve momentum in building up one's list of contacts and starting the sales cycle.
- One year to demonstrably fail to achieve results.
- One year for HR to convert the salesman from non-performing to "no longer with the firm."

Given that it takes four full years to go through this cycle, you could see a salesman with an impressive resume of 3 ~ 5 year stints at name brand companies with impressive sales to their credit... and not learn, from their resume, that their sales skills are atrocious and their successes were actually the result of products which could be sold solely on the strength of their quality/marketing/brand even with the salesman impeding customer perception of all three. For example, Jason's Frank probably fit into this category.

Paul nonetheless believes that strictly depending on inbound marketing restricts your business to plucking the low-hanging fruit, and that going solely after low-hanging fruit is not a great business strategy. (This would be an excellent opportunity to whip out the above quadrant map, by the way: if you're in either of the B2C quadrants, congratulations, low-hanging fruit *is* your business strategy, and you can be quite successful with it. Ditto 37Signals and many similar cheaper B2B SAAS apps, by the way.)

The point of speaking to customers, which is all sales is, is to reach a mature understanding of what your client values the most. Then, and only then, you offer that to them.

Engineers routinely suck at doing demos for customers because they demo every bit of the interface, from start to finish, going through all the little knobby settings bits. Nobody cares about these things. Demo the bits of the software your customers find interesting. When you email customers, talk about the bits of the software your customers find interesting. When you blog about the software, blog about the bits of the software your customers find interesting. But above all things recognize that your customers find *themselves* interesting and the software *boring* unless the software makes a demonstrable improvement in their lives.

Quick tip when speaking to customers: "The most interesting people you will ever meet are the people who are most interested in you." The average sales rep spends about 47 seconds in talking about customers prior to talking about the solution. That number should be increased, *greatly*. On the plus side, because the industry is so comprehensively screwed up here, even minor improvements provide an opportunity to surprise and delight customers. "Goodness, the CEO of that company spent *five whole minutes* talking to me about how we go about our business! *They really care!*"

(Sidenote: This also goes if you're selling yourself as an employee to a company. If you're in an interview and talking about the employer, you are probably going to get hired. For reasons beyond my ken they don't teach this lesson in college.)

Key takeaway: The quality of your dialogue with customers is directly proportional to the quality of the customers you will acquire. If you understand their needs better, you will close bigger deals with happier customers who consume less resources.

Customer faith in your product as a solution to their problem is directly proportional to how well they believe that you understand their problem. This again counsels spending more time talking to them and asking perceptive questions, then repeating their own language right back at them. If they call it a foo, you call it a foo, even if internally everyone knows it is "really" a bar and, after all, it implements IBarable in the source code. Relatedly, you cannot *tell* your way into a dialogue with the customer, you can only *ask* your way in.

Customers have DNA: Drivers, Needs, and Aspirations. You should be capturing your understanding of these as you talk to customers, or you aren't learning what you need to learn to bring the customer and the firm to a mutually satisfactory relationship.

Some factors to consider

- Customer needs
- Timescale for implementation
- Scalability
- Integration with existing systems/processes
- Affordability
- Results

Customers have many priorities:

- Ego (underrated by engineers in my opinion... even those who own iPhones because they're worth owning iPhones)
- Perceived gain
- Sense of belonging (“Nobody ever got fired for...”)
- Security
- Ease of use

If you think of a funnel of concerns prior to sale, starting at the top:

- The experiences customers have right now.
- Their business case or project which may benefit from your solution.
- The utility your solution can offer. (Note to engineers: many of you stop inquiring here. That is a mistake.)
- Options they have competing with your solution.
- What the company values in terms of outcomes, drivers, etc.
- Resources they have to solve the problem (i.e. talk budget *last*, not first)

Founders have an advantage in doing sales, even if they're engineers who think they can't do sales.

Nobody understands the problem domain like a founder, nobody understands the solution's capabilities better than a founder, and nobody else can say “You can trust me on this because I build the bloody company” or inspire the same degree of emotional response in a customer that talking to a founder provides. (This is so true in my personal experience. I hear, over and over again, from my customers that they trust me because they know they get an answer “straight from the top” when they send me an email. I hear that most often from people who *never emailed me in the first place*, or who emailed me with issues that could be easily dealt with by Level 1 CS. This is one compelling reason why I don't outsource customer service.)

Rob Walling

Rob writes [Software By Rob](#), runs a stable of small software businesses as a one-man show (including, most amusingly to me, selling beach towels — he focuses just on software these days), and wrote [a book I really enjoyed](#). (Flip to the chapter about Virtual Assistants, it is worth the price of the book and then some.) The [slides](#) and [outline](#) for his talk are online.

After review of statistics generously donated by several software companies, Rob is of the mind that the most important goal of your website is to bring people back for additional visits, because the conversion rate of your website is much, much higher among engaged visitors than it is among first-time visitors. I didn't write down the exact numbers, but I recall him citing DotNet Invoice (“written in Ruby on Rails... no, just kidding”), CrazyEgg, and the beach towel site among a few others.

Interestingly, a few of the examples had higher average order values for returning visitors, too.

I only investigated this for about 30 seconds, but I have lower conversion rates to the online free trial of my software from returning visitors than first time visitors. That actually isn't contrary to his thesis, though — it is an artifact of having a very scalable SEO strategy and customer acquisition highly dependent on high-converting organic and AdWords landing pages. This is one of the threats of making decisions based on shallow understanding of what Google Analytics tells you is the “truth” of your business. (One minor surprise: about 60% of my customers who convert from the free trial convert within 3 hours of starting it. Nail that first run experience and nail it hard, guys.)

“The ineffective marketer asks you to buy too soon.” A customer on their first visit could have multiple reasons why they are not currently prepared to buy your software:

1. They don't have enough info about it.
2. They don't trust you. (Oh goodness, this topic is worth a book.)
3. They don't have money to buy it right now. (A particular issue for a \$300 invoicing software.)
4. They don't need the software right now.
5. They're never going to buy this.

You can recover from all the issues but #5 if you establish a relationship with them. Permission-based email marketing is the best way to do this, bringing them back to your website via a communication channel everyone has and uses. Repeated, trust-building contact with them will gradually wear down their resistance to parting with their cold-hard cash.

Email is far superior to competing methods of repeatable contact with customers — such as Twitter, blogs, RSS feeds, or your favorite social network — because it allows “personalized broadcasts”, where you can scalably communicate with an arbitrarily high number of people while also making it feel like everyone is getting individualized attention. (Not nearly enough people use this to its maximum potential, particularly with SaaS. Free tip from me: note what search term sent them to your website, put a variation of it in the subject line of the email, watch your open rate go through the freaking roof.)

To get the maximum value out of emailing your customers:

1. Have a killer landing page laser targeted on the “give us your email in return for this immediate benefit to you” call to action. Give them a whitepaper or similar resource in return for the email. It greatly, greatly will increase your conversion rate. (Rob is not a fan of free-to-use trials. That is an interesting idea to me, as I come from a “of course the software is free to try out” background. He has achieved truly eyepopping conversion rates to an email submission — 40%+ on some pages. I can't get that to my free trial, which also requires an email address, and I have pages which are very, very compelling at “selling” that to customers who were looking for exactly what the free trial offers.)
2. Give something away in return for the email address. See the slides for specifics, but this has a monumental ROI. (I have some confidential numbers from consulting clients. Suffice it to say this is right on the money.) Interestingly, if you just call a white paper a “report” it has higher perceived value. (News you can use!)
3. Set follow-up for emails. [MailChimp](#), my mail provider of choice, calls this an autoresponder sequence. If you take one thing from this presentation, learn and use autoresponders. They will change your email marketing life.

Some excellent tactical suggestions:

- Spam: Don't spam! Don't mistakenly get classified as spam by hitting filters. You can pre-test your emails against common filters, like SpamAssassin, to reword against accidental false positives. (A long time ago I worked on anti-spam systems for a living. Oh, the hilarious stories I could tell you. Suffice it to say that a home builder might want to avoid promising that

they could “increase the size [of your patio]“.) Incidentally, MailChimp will do this for you for a nominal per-email fee.

- Some words kill open rates if you use them in subject lines. Reminder, Special, % off, and “Help [us]...” are all culprits.
- From Name: your own personal name is best, and role names are a bit worse, company name is a bit worse, and webmaster@example.com is worst of all.
- **All advice on emails can be A/B tested!**

Peldi (Giacomo Guilizzoni)

Peldi runs [Balsamiq](#), a small software company which has become practically synonymous with low-fidelity mockups for software. He wrote the [best blog post on the Internet](#) about seeking coverage for your startup. You should execute on it.

The theme of the talk was worries: worries you shouldn’t have, worries you will have, and what keeps Peldi up at night these days.

Things you should not be worried about:

- Asking customers to pay you money. (Both Peldi and I were shy about this prior to starting, apparently, because we had the genetic flaw all engineers have that says that there might be a bug and, therefore, it is too early to accept appreciable amounts of money for the product. I promise to do my part and say “Charge more” to any person that will listen to me until we have cleansed this from our collective gene pool.)
- Pirates. (Not an issue in practice, you can pretty much fire-and-forget a [low-pain DRM system](#), and since if you’re smart [you’re doing web apps anyway](#) this is disappearing from the radar screen entirely.)
- Picking a niche which is too small. (Peldi put up a screenshot of [Bingo Card Creator](#) here, to the widespread amusement of the crowd. I later heard more than one attendee say that they were under the impression it was “a funny Italian joke” until I gave my lightning talk. That actually worked out great for my talk’s reception.)

Early worries in the lifecycle of his company:

- When to make the jump from full-time employment to starting your own business. If I recall the talk correctly, Peldi scrimped and saved for a while, sold his shares of Adobe, and emptied his IRA so that he would have a year of runway in hand prior to going full-time on Balsamiq. He also had written code on nights and weekends for several months, so he was not doing a standing start. (And his rise was truly meteoric after that — a combination of product quality and being a marketing genius. If you don’t understand why he is a marketing genius, go back and read that post I cited earlier.)
- Work/life balance. Regrettably, you’re going to have to get your family on board with losing you for a few months while you disappear into the Bat Cave and hammer out your software. (This is my one bone to pick: no, you do not. You can virtually always stretch out your schedule.) Peldi then delivered the best line of the conference: “If you work while they sleep, they won’t know that you’re ignoring them.”

Many people are under the impression that Balsamiq was an instant, overnight success, largely based on Peldi’s embrace of radical transparency and publishing revenue numbers. Peldi had a series of very interesting graphs: the first shows the hockey stick revenue curve that competitors salivate over. Then it zooms out to include the time while the software was getting written (i.e. no sales). Then it zooms

out to include his previous career at Macromedia/Adobe where he learned to polish software to perfection in the way that separates Balsamiq from its many, many competitors. Then it zooms out to include his programming career starting at age 12. Overnight success only took thirty years. In a conference with many profound insights, I think this was one of the best ones.

Let me give you an example from my own experience: I recently received a wire transfer from a consulting client, for an amount of money which would be unexceptional to some people and which is staggering for me — more than I've ever had accessible in my entire life, by a factor of lots. It was for two weeks of work and was roughly equivalent to what I used to make for six months at my previous day job. The story for how I got to do those two weeks of work starts out with participation on the [Joel on Software discussion boards](#) some four and a half years ago. I started dabbling in the topic the work covered almost six years ago. The focus on communication which made it possible to sell the client (and eventually, their team) on the importance of the work getting done can be traced back probably to middle school (when I took up public speaking to get over a speech impediment). You literally needed an *entire lifetime* to prepare for what you're doing today.

Back to Peldi: don't be afraid of an inability to deliver. You're a talented engineer, you'll figure it out. If you fall in love with the problem and can discuss it with boundless energy, then every interaction with customers will make your product better. At early stages in the product lifecycle, feedback from customers is a much better thing to capture than mere money. (Peldi gave away licenses like they were candy during the first few years. Again, read the above blog post, in addition to getting great feedback this was a great PR and linkbuilding hook.)

Peldi had an extended discussion of how you can use Obama's debate script as a model for addressing your fear of confrontation with customers. While I'm not a fan of his politics, I agree, he is an excellent communicator. There was a flow-chart involved, but the core is:

- Thank the customer for taking time out of their busy life to communicate with you.
- Empathize with them.
- Apologize if they're ticked off, even if they have no right to be. (Oh, crikey, how much do [I agree with this.](#))

Lightning Talks

All of the Lightning Talks were delivered by non-professional speakers. They followed a murderously difficult format: you get 15 slides which get auto-advanced every thirty seconds, and that's it. You might think that means you can get away with slapping together something in an hour: oh no. In discussions with my fellow lightning talkers, we agreed that there has been something of a "lightning talk arms race": the two talks I was most impressed by took over 24 hours to prepare, and mine took at least twelve solid hours over two months, with probably half of that being rehearsal until I could literally count out thirty seconds with a prepared spiel delivered in a voice other than my own.

I'm going to refrain from talking about my own lightning talk. Why? First, it relies on a "reveal" for a great deal of the impact. Second, it was, far and away, the best speech I've ever given (out of thousands — some people play sports, I did public speaking). The combination of heartstopping terror and the crowd reaction produced a strange alchemy which made the talk as delivered *a lot* better than the videos I made of myself performing it in a room for my webcam.

I'm told it was recorded and the video will be made available to the public. I'll tell you when it is ready. I promise you, you'll enjoy that a lot more than you'll enjoy me posting the slides and a video of me saying virtually the same words to a wall.

As to the lightning talks other than mine, a combination of heartstopping terror about my talk and the exigencies of the format mean that my notes on them are woefully inadequate. I expect that a few of the lightning talks will eventually be put online at some point. Those videos will do them better justice than my hazy recollections, so I recommend waiting and seeing them.

Eric Sink

Eric presented the nuts and bolts of what happens if your company (or, in his case, a product thereof) gets acquired by BigCo (in this case, Microsoft). Short version: a lot of pain and suffering, followed by a check so large that the VP of his community bank said putting it in his account would throw their company into absolute havoc because of the impossibility of putting so much money to work. Let me emphasize the pain and suffering part, though. (The low point: when the deal had consumed his waking life so much that Eric was reduced to taking a phone call from his lawyer while in church.)

One of the weirder points for Eric was that, with a cast of literally dozens working the Microsoft side of the deal, he never met or learned the identity of the person who actually made the decision to buy the product. In a gang of people with confusingly similar job titles and descriptions, it seemed that “nobody was in charge.” The deal was lead by a particular point of contact — we’ll call him Fred — who, as a trained negotiator whose full-time job is assimilating biological and technological distinctiveness, was a thousand times more prepared than any entrepreneur will ever be.

The process involves a long rigmarole of procedural steps. You’ll want to see the full talk for the breakdown, because I’m hazy on it, but it involved multiple levels of due diligence, getting signed agreements covering IP rights from every person who had ever put a finger to a keyboard within a meter of the source code (*cough* start work on this today if you want to sell *cough*), etc etc.

While many entrepreneurs obsess about The Number, what you actually get is The Deal, which ended up being 150 pages of bullet points *all individually negotiated* covering every aspect of the transfer.

This included guarantees of treatment for employees who were relocating with their product from central Illinois to Microsoft, all manner of indemnifications, tax treatment of the deal, escrow (code and otherwise), the exact structure of compensation for the deal, the terms of the NDA covering material terms of the deal, etc etc.

The deal nearly fell through at several points, over contentious negotiating issues. Nearly might not be strong enough: it was *dead*. The deal was off, and the other side went to total radio silence for a period of weeks. This was necessary to get the best achievable outcome for Eric and the team. Eric is of the opinion that you need to walk away about twice: less and you’re getting screwed, more and you’re buying more heart attacks than the marginal improvement in terms is worth. (One friend of his — in the elite fraternity of “people who have sold a company”, whose stories “you only become privy to after you have joined the club” — had to walk away *five times*.)

One interesting point raised in discussion was the importance of having multiple interested buyers so that you can play them against each other. (Michael Pryor from Fog Creek is especially articulate about this one. The short version: if your BATNA is “get bought by the other guy”, you have more leverage than if your BATNA is “you don’t get acquired, six months of work flushes down the drain, and the company potentially withers.” The long version: ask Michael.)

Dan Bricklin

So here is a generation gap question for you: I didn’t know who Dan Bricklin was, and someone said “Well, duh, he’s the guy who wrote VisiCalc.” I then had to Wikipedia [VisiCalc](#) because I had never

heard of it. For those of us who were born in the eighties or later: he invented spreadsheet software. Yowza.

In addition to a hundred and one precious anecdotes about inventing spreadsheet software while a poor grad student (my favorite: the development machine and sole source repository, which cost the team's life savings, being protected from a plumbing problem by constant vigilance with mops), there were a couple of generally useful takeaways:

- Customers have jobs that they need to get done. Identify those jobs. Build **general purpose** software which allows them to get the jobs done.
- The perceived value of general purpose software, which solves a variety of current and anticipated needs of a customer, is higher than that of software which solves one specific pain point. (I respectfully disagree, but then again, I didn't invent the spreadsheet.)
- Software which the customer can customize to their particular needs allows "tailoring at the ends."
- The two-week payback rule: software sells itself if you can reliably demonstrate that your product will recoup the purchase price within two weeks. VisiCalc (spreadsheet on a mini-computer) was so disruptively better than using a time-sharing system to do calculations that you could buy the software and the system to run it for just half of one month's metered use of the big mainframe you leased time on to run your TPS reports. That contributed to it selling 700,000 copies, back when distribution channels for software were stone age compared to the alternatives we have today. (My unconscious brain was virtually screaming: "You can sell software *without a website!*" when I saw the magazine ad slides.)
- Games and relative cheapness were very important in driving new technology adoption. Note that this, and the "general purpose applications win" rule, are demonstrated in abundance by the runaway success of the iPhone.

Derek Sivers

Derek had originally planned on delivering a talk about various profit models for software companies (spiritually similar to [this talk](#)). I met him for the first time at a speaker's dinner. During the course of dinner, one topic frequently returned to was why we do what we do. Derek told a very moving story about how he sold CDBaby, his labor of love for many years, and someone (I think it was Rob Walling) suggested that he do that as a talk instead. He ended up doing so.

Where to start:

Derek originally got into CDBaby to sell CDs of his own band's songs. Back at the dawn of the Internet, getting a merchant account for selling CDs was a multi-month process, with plenty of administrative hassle and pain involved. Derek persevered and, with no programming background, managed to hack together enough CGI scripts to get a buy now button on his website in a mere three months. (And again I bless my lucky stars that I had Paypal and [e-junkie](#).)

After he started selling CDs, some of his friends with bands said "Hey man, that's really cool. Can you sell my CDs, too?" So he'd hack the scripts again and then he was selling another CD. Eventually, word of mouth had gotten to the point where his band's website was colonized by other bands who lacked a good way to sell their CDs online, and CDBaby was born. [His blog](#) covers this story in detail.

Derek was once interviewed by NPR and, when asked about his exit plan, said that long after the sun had set on CDs and when he was old and grey, he'd still be shipping CDs to the last aging person who swore that the CDs they played when he was young were far superior to that newfangled hologram 7D

garbage that passed for music in 2050.

This turned out not to be the case: after a major rewrite of the CDBaby source code, where Derek achieved pretty much every technical goal he had for the site (and had paid back years of code debt), and after changing his management style such that he became totally superfluous in day-to-day operation of the company, Derek became an absentee owner of the company he founded. Some people might view that as a wonderful outcome. It turns out that Derek's employees viscerally hated him for it: convinced that they were the true bearers of the CDBaby vision, rather than the unseen owner who many of them had never even met, they began organizing what could be described as a corporate coup.

When Derek found out about this (oh, the joys of reviewing MP3 recordings of meetings at midnight), he realized that he was well past the point of no return with regards to employee loyalty. So he turned Apache off and started composing an emailed pink slip for the entire staff. No, *really*. Luckily for all concerned, he eventually decided to sleep on it, and came back to advice he had received from Seth Godin: if you are no longer passionate about the problem, you owe it to your customers to sell to someone who is. Apache went back on and the email was not sent.

Derek pursued a few options for getting rid of the company. One was to simply give it away: he actually pursued pulling off a Willy Wonka and putting five golden tickets in CDs delivered by his company, with the five holders invited to present their visions for the future growth of the company, and the winner (as decided by a vote of customer musicians) receiving the entire company. Due to fortunate advice from friends regarding the desirability of being sixty and having no money, Derek threw away the golden tickets he had gotten printed and decided to do the more usual thing and just sell.

The sale came with a wrinkle: Derek first transferred the company into a [charitable trust](#), then sold it, meaning that he is forced to draw 5% of the sale price per year, and when he passes away the principal and investment gains (if any) of the trust will be distributed to the charitable causes of his choice.

Derek claims this is mostly not for altruistic purposes: he asked his lawyer how to achieve "I want to have a stable income until I die, and that is all I need", and the charitable trust was an attractive option.

Derek mentioned that having competing bidders was wonderful, since they could be played against each other. This helped him achieve some goals he had in selling the company:

- I'm out. (Founder participation as a consultant or employee for a defined term is a frequent requirement of sales. Derek took that option off the table entirely as a precondition of sale.)
- I keep my database. (A decade of running essentially a marketplace between music sellers and music buyers gives Derek a powerful foundation for future endeavors aimed at helping musicians, his perennial concern.)
- I keep helping musicians. (The non-compete was narrowly tailored so that he would not sell CDs again, but could continue participating in the music industry. Sometimes non-competes are very broadly drafted — I've heard possibly apocryphal reports from one that, boiled to its essence, barred a technical founder from programming for the duration that a large company with a small-sounding name continued selling operating systems.)

Joel Spolsky

I'm pretty sure most people know [Joel](#) co-founded [Fog Creek](#) and [Stack Overflow](#).

Continuing the theme of the later part of the conference about big picture questions like "Why?", Joel's presentation was probably the bravest speech I've ever heard. Theoretically, it was about the story of transitioning from a low-growth (ish) software business like Fog Creek to a high-growth VC-funded

shoot-the-moon attempt like StackOverflow. However, it dwelled on some intensely personal questions about leadership, philosophy, and intra-team conflict, and I don't know if I could have discussed them at that level in public.

Let's see: the brief takeaway on seeking investment is that if you've got a successful track record as a well-known CEO and a good idea which also has demonstrable traction, raising capital is fairly easy (Joel and company were practically booked solid when they went to Silicon Valley to meet with VCs). However, it still has some of the industry seediness. One VC who they met with, who ended up not investing in the company despite early promise, later tried his darndest to cause a rift between Joel and his co-founder (Jeff Atwood) so that he could pick up the pieces. Another VC was quite interested in StackOverflow, provided it was CEOed by anyone other than Joel.

Interference from VCs aside, they eventually got signed by Union Square Ventures, who have treated Joel and company fairly. The cultural gap between a self-funded software company and the VC world has caused a few disconnects (for example, at one point they found that StackOverflow had \$200,000 lying around that they had forgotten to disclose based on revenue, and they asked what corporate structure should receive it or whether they could licitly award it to employees: the VCs were totally uninterested in \$200,000, and rather amused at the notion that a company they had funded could actually have had a profit from operations at the stage they were at).

StackOverflow has a very different corporate culture than Fog Creek, which Joel describes as a software company founded with the goal of becoming an "intentional community" whose mission in life was providing programmers a great place to work. It was explicitly analogized to a kibbutz, with collective ownership, the profit being redistributed among employees, and decisions specifically made to promote community (one which stuck with me was the huge importance placed on breaking bread together on a daily basis).

The culture involves decision-making by consensus. For a variety of reasons, StackOverflow does not operate in this fashion. This led to some contentious battles between Joel and Jeff, exacerbated by the aforementioned evil VC, who tried to split the founders to pick up the company on the cheap. Among other things, after frequent screaming and crying, Joel figured that his traditional style of hiring smart people and letting them get things done wasn't working, and he called in a CEO coach for assistance. He thought he'd get management advice on how to better run meetings and write org charts and do the things that CEOs do.

Instead, he got corporate psychoanalysis, and was introduced to the notion that the CEO's job is not to *manage* the company, but to *lead* it. This was particularly important as he was CEOing two companies simultaneously. Once he started accepting that "I am the CEO, I have heard your viewpoints, *we do it my way and that is final*" was also a valid managerial style, conflict in the team ironically decreased greatly.

On the question of why we do what we do: Fog Creek is the company Joel designed to have a great place to work. StackOverflow, on the other hand, is targeted at contributing one significant thing to the world: fixing the broken style of asking expert questions on the Internet. They want to bring the improvement StackOverflow brought to programmers to the wider community (and communities) of interest who Google questions which have canonical good answers and find a SERP stuffed with 10 year old VBulletin threads where the answer is on page 7 in between a cat photo, a flame war, and the announcement of the forum's guru's goddaughter's wedding.

Joel would prefer to work at Fog Creek, but thinks he has an ethical responsibility to make StackOverflow succeed first, because — irrespective of Fog Creek maximizing his personal enjoyment — StackOverflow has the potential to contribute a large social good to the world at large, and he views this as ethically mandatory given the opportunity. He contrasts this to the 37Signals-esque notion of

being content to run a small business because it allows you financial success and the lifestyle you enjoy (although he was quick to note that, while saying this, 37Signals gave the world Rails and teaching, so their own ethical bases are covered).

For an example, see CraigsList, which has moved a huge amount of wealth from newspapers to the posters of classified advertisements by making classifieds free. CraigsList is run with a goal other than profit, and would instantly be a gigantic company if they charged for ads in categories other than the ones they do now (which they do as a spam-control device). Joel argues that Craig *should* do this, because the massive wealth transfer his business enabled has redistributed the producer surplus from newspapers (which once used it for a socially beneficial purpose: underwriting unprofitable but socially beneficial local investigative journalism) to a consumer surplus for “people selling couches, landlords, and pimps.” His point was that, when we know the trade-off our businesses are making, we have an obligation to choose the results wisely.

I am perhaps not doing this point justice: the two minute thumbnail sketch of the argument I heard Joel deliver *profoundly* altered my perception of where my company fits into the larger tapestry of my life.

I would previously have characterized, e.g., the decision to keep the company small as being essentially a personal aesthetic choice with no moral consequences, akin to picking colors for one’s bedroom. I will be digesting the implications of this for a while, but I think I am now convinced that that is probably untrue.

A sidenote: since, pace Seth Godin, the cost of producing software is cratering and the competition is exploding, Joel is gradually coming to the belief that the value in a software company isn’t the code, it is the community. The thing StackOverflow had to offer VCs wasn’t the code (which, to quote what has become an in-joke among HN readers, “could be written in a weekend”), it was the “large number of engaged users.” Microsoft estimates there are 9 million programmers in the world. StackOverflow has upwards of 8 million monthly uniques. They’ve become the canonical go-to place for answers in the programming niche, and have dreams of becoming the canonical go-to place for answers for all expert queries which have canonical right answers, from tax advice to math theorems to gardening and many points beyond.

Commitment To Community & Disclaimer

[\(Back to top.\)](#)

Like I mentioned earlier, I *strongly* recommend you attend Business of Software. Even at close to 10,000 words, this post captures a bare fraction of the value I got out of going to the conference. I understand that both the ticket and the ancillary costs of attending are quite high for bootstrapped startups. Trust me on this one: a plane ticket from Japan to Boston and four days in a hotel wipe out nearly a month of revenue for me — and it was worth every penny and then some.

Neil Davidson, the conference organizer, was very generous with [giving away tickets](#) to the event to deserving bootstrappers. In the spirit of the overwhelming generosity of the community and the thoughts sparked by talk of our ethical responsibilities, I will likewise make an opportunity for at least one marginal person to go next year. Watch this space — I’ll hammer out specifics closer to the actual event.

Now, the obligatory disclaimer: have never accepted advertising on my blog and don’t feel like starting anytime soon. I was given a free ticket to the conference in return for presenting my Lightning Talk at it, and after winning the competition for best Lightning Talk (which I literally did not know existed until five minutes before it was announced that I won it), I received a Kindle and an invitation to a very nice dinner. I’m easy to impress, but not that easy: my effusive enthusiasm for this conference comes

from it being one of the highlights of my professional career, not because of compensation.

Posted in [Uncategorized](#) | [31 Responses](#)

[Security Lessons Learned From The Diaspora Launch](#)

By [Patrick](#) on

Last week, [Diaspora](#) — the OSS privacy-respecting social network — released a “pre-alpha developer preview” of their [source code](#). I took a look out it, mostly out of curiosity, and was struck by numerous severe security errors. I then spent the next day digging through their code locally and trying to get in touch with the team to address them, privately. In the course of this, I mentioned obliquely that the errors existed on Hacker News, and subsequently was interviewed by [The Register](#) and got quoted in a couple of hundred places.

The money quote most outlets went for was:

The bottom line is currently there is nothing that you cannot do to someone’s Diaspora account, absolutely nothing.

I’d like to back up that contention, now that it is safe(r) to do so.

Reporting security bugs is a funny business: any description of the error sufficient to resolve it is probably sufficient to create exploit code. This is why I was fairly circumspect about the exact mechanism for the errors, and why Steve Klabnik also mostly [declined to give specifics](#) when describing the state of the codebase. Since the specific errors I reported are now patched, I’m going to disclose what they were, so that budding Rails developers who care about security do not inadvertently give attackers the ability to do anything they want.

By the way, if you’re looking for Rails security advice, I recommend the [official guide](#) and the [OWASP list of web application vulnerabilities](#), which would have helped catch all of these. Web application security is a **very deep topic**, and often involves unforeseen circumstances caused by the interaction of complicated parts which are not totally under the developers’ control. That said, **nobody** should be making errors like these. It hurts us as developers, it hurts our ecosystem, and it endangers our users in spite of the trust they have put in us.

I found somewhere in the ballpark of a half-dozen critical errors — it depends on how you count pervasive mistakes that undermined virtually every class in the system. There were three main genres. All code samples are pulled from Diaspora’s source at launch, were reported to the Diaspora team immediately, and have been reported to me as fixed.

Authentication != Authorization: The User Cannot Be Trusted

Code:

[view sourceprint?](#)

```
1.#In photos_controller.rb
2.def destroy
3.    @album = Album.find_by_id params[:id] # BUG
4.    @album.destroy
5.    flash[:notice] = "Album #{@album.name} deleted."
6.    respond_with :location => albums_url
7.end
```

This basic pattern was repeated several times in Diaspora's code base: security-sensitive actions on the server used the params hash to identify pieces of data they were to operate on, without checking that the logged in user was actually authorized to view or operate on that data. For example, if you were logged in to a Diaspora seed and knew the ID of any photo on the server, changing the URL of any destroy action from the ID of a photo you own to an ID of any other photo would let you delete that second photo. Rails makes exploits like this child's play, since URLs to actions are trivially easy to guess and object IDs "leak" all over the place. **Do not assume** that an object ID is private.

(There is a second error here, by the way: the code doesn't check to see if the destroy action is called by an HTTP POST or not. This means that an overenthusiastic browser might follow all links from a page, including the GET link to a delete action, and nuke the photo without any user action telling it to do so.)

You might think "Surely Diaspora checks to see if you're logged in, right?" You're right: they use Devise, a library which handles **authentication**, to verify that you can only get to the destroy action if you're logged in. However, Devise does not handle **authorization** — checking to see that you are, in fact, permitted to do the action you are trying to do.

Impact:

When Diaspora shipped, an attacker with a free account on any Diaspora node had, essentially, full access to any feature of the software vis-a-vis someone else's account. That's **pretty bad**, but it gets even better when you combine it with other errors.

How to avoid this:

Check authorization prior to sensitive actions. The easiest way to do this (aside from using a library to handle it for you) is to take your notion of a logged in user and only access members through that. For example, in my software, any action past a login screen has access to a `@user` variable. If an action needs to access one of their `print_jobs`, it calls `@user.print_jobs.find(params[:id])`. If they have subverted the params hash, that will find no `print_job` (because of how associations scope to the `user_id`) and they'll instantly generate an ActiveRecord exception, stopping any potential nastiness before it starts.

Mass Assignment Will Ruin Your Day

Code:

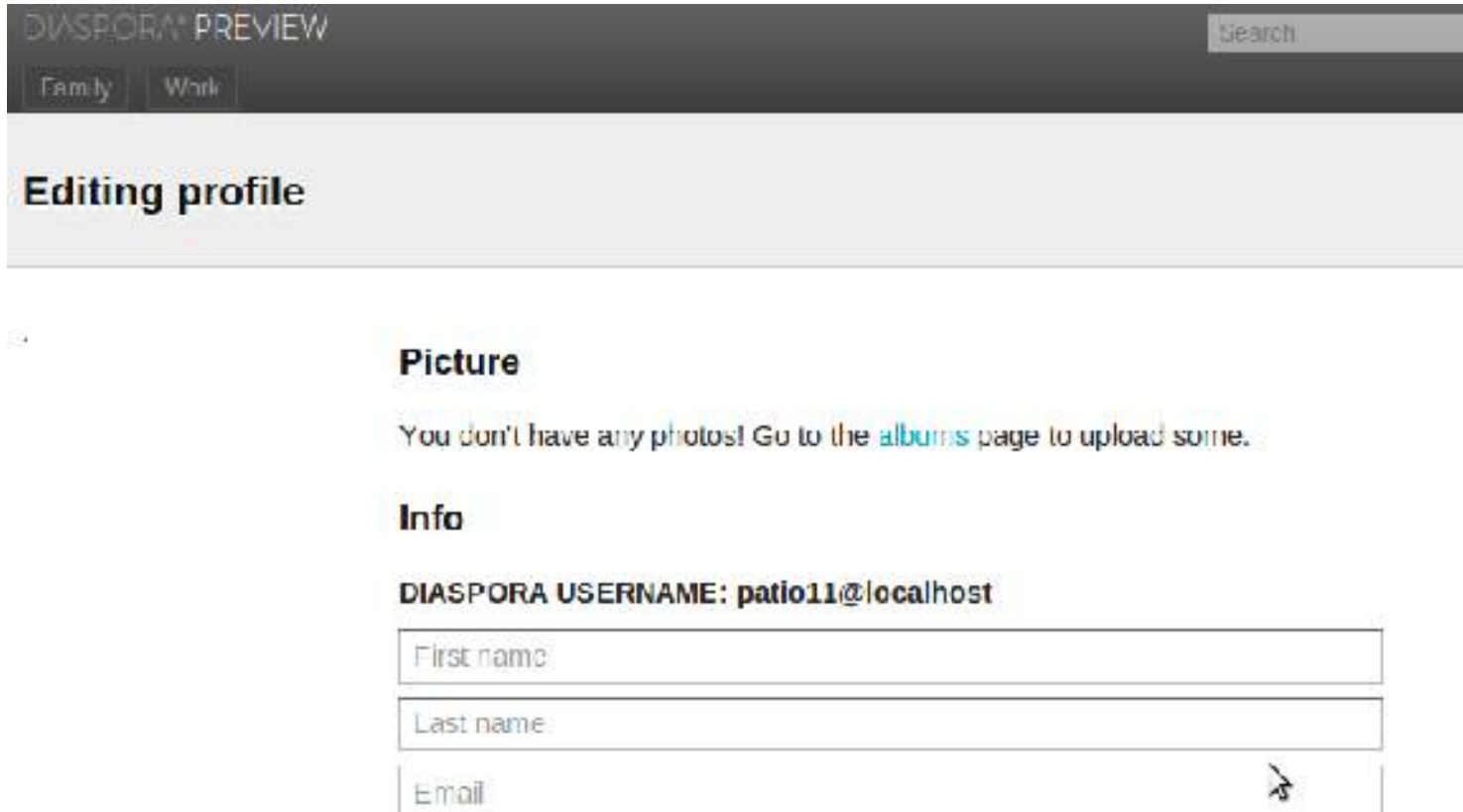
[view sourceprint?](#)

```
01.#users_controller.rb
02.def update
  03.  @user = User.find_by_id params[:id] # <-- uh oh, no auth check
  04.  prep_image_url(params[:user])
05.
  06.  @user.update_profile params[:user] # <-- pass untrusted input
      to @user then...
  07.  respond_with(@user, :location => root_url)
08.end
09.
10.#user.rb
11.def update_profile(params)
  12.  if self.person.update_attributes(params) # <-- insert input
```

```
directly to DB
13. ...
14. end
15.end
```

Alright, so we know that if we forget authorization then we can do arbitrary bad things to people. In this case, since the user update method is insecure, we can meddle with their profiles. But is that all we can do?

Unseasoned developers might assume that an update method can only update things on the web form prior to it. For example, this form is fairly benign, so maybe all someone can do with this bug is deface my profile name and email address:



The screenshot shows a web interface for editing a profile. At the top, there's a header with 'DIASPORA PREVIEW' and a search bar. Below that are tabs for 'Family' and 'Work'. The main heading is 'Editing profile'. Underneath, there are three sections: 'Picture', 'Info', and 'Email'. The 'Picture' section says 'You don't have any photos! Go to the [albums](#) page to upload some.' The 'Info' section displays 'DIASPORA USERNAME: patio11@localhost' and has three input fields: 'First name', 'Last name', and 'Email'. The 'Email' field has a small icon of an envelope.

This is **catastrophically wrong**.

Rails by default uses something called “mass update”, where `update_attributes` and similar messages accept a hash as input and sequentially call all accessors for symbols in the hash. Objects will update both database columns (or their MongoDB analogues) and also call `parameter_name=` for any `:parameter_name` in the hash that has that method defined.

Let's take a look at the `Person` object to see what mischief this lets us do. (Right, instead of updating the profile, `update_profile` updates the `Person`: Diaspora's internal notion of the data associated with one human being, as opposed to the login associated with one email address (the `User`). Calling something `update_profile` when it is really `update_person` is a good way to hide the security implications of code like this from a reviewer. Names matter — make sure they're accurate.) What methods and fields do you expose...

[view sourceprint?](#)

```
01. #Person.rb
02. class Person
03.   ...
04.   key :url, String
05.   key :diaspora_handle, String, :unique => true
06.   key :serialized_key, String #This is your public/private
    encryption key pair. OOPS.
07.
08.   key :owner_id, ObjectId #You don't want me changing this
    one either...
09.
10.   one :profile, :class_name => 'Profile'
11.   many :albums, :class_name => 'Album', :foreign_key =>
    :person_id
12.   belongs_to :owner, :class_name => 'User' #... because it lets
    me own you! Literally!
13.
14. end
15.
16. #User.rb
17. one :person, :class_name => 'Person', :foreign_key => :owner_id #Oh
    dear.
```

This is painful: by changing a Person's `owner_id`, I can reassign the Person from one account (User) to another, allowing me to both deny arbitrary victims from their use of the service and also take over their account, allowing me to impersonate them, access their data at will, etc etc. This works because *one* in MongoDB picks the first matching entry in the DB it can find, meaning that if two Person have the same `owner_id`, your account will non-deterministically control one of them. So I'll assign your `Person#owner_id` to be my `#owner_id`, which gives me a fifty-fifty shot at owning your account. If that is annoying for me, I can always assign my `Person#owner_id` to have some nonsense string, de-linking them and making sure `current_user.person` finds *your* data when I'm logged in.

But wait, there is more!: Note the `serialized_key` column. Can you guess what that is for? Well, if you follow some spaghetti in the User class, that is their serialized public/private encryption key pair. You might have heard that Diaspora seeds use encryption when talking between each other so that the prying eyes of Mark Zuckerberg can't read your status updates. Well, bad news bears: the attacker can **silently overwrite your key pair**, replacing it with one he generated. Since he now knows your private key, regardless of how well-implemented your cryptography is, he can read your messages at will.

This is what kills most encryption systems in real life. You don't have to beat encryption to beat the system, you just have to beat the weakest link in the chain around it. That almost certainly isn't the encryption algorithm — it is some inadequacy in the larger system added by a developer who barely understands crypto but who trusts that sprinkling it in magically makes it better. Crypto is not soy sauce for security.

Is this a hard attack? **No.** You can do it with no tool more complicated than Firefox with Firebug installed: add an extra parameter to the form, switch the submit URL, own any account you like. It took me **two minutes** to find this vulnerability (I looked at the users controller first, figuring it was a likely place for bad stuff to happen if there was bad stuff to be found), and started trying to get the word to the Diaspora team immediately. It literally took longer to get Diaspora running than it took to create a

script weaponizing this.

Steps to avoid: First, fix the authentication. That won't prevent this attack, though — I can still screw up *my* Person by changing its `owner_id` to be yours (and do this an arbitrary number of times), virtually guaranteeing that I can successfully disassociate your account from your person.

After you fix authentication, you need to start locking down write access to sensitive data. Start by disabling mass assignment, which should be off in an public-facing Rails app. The Rails team keeps it in because it saves lines of code and makes the 15 minute blog demo nicer, but it is an easy security hole virtually anywhere it exists. Consider it guilty until proven innocent.

Second, if your data store allows it, you should explicitly make as much as feasible unwritable. ActiveRecord lets you do this with `attr_readonly` — I'm not sure whether you can do it with MongoMapper or not. There is almost certainly **no legitimate reason** for `owner_id` to be reassignable.

NoSQL Doesn't Mean No SQL Injection

Code:

[view sourceprint?](#)

```
1. def self.search(query)
  2.   Person.all('$where' => "function() { return
    this.diaspora_handle.match(/^#{query}/i) ||
      3.     this.profile.first_name.match(/^#{query}/
        i) ||
      4.     this.profile.last_name.match(/^#{query}/i
      ); }")
5. end
```

Diaspora uses MongoDB, one of the new sexy NoSQL database options. I use a few myself. They have a few decades less experience getting exploited than the old relational databases you know and love, so let's start: I claim this above code snippet gives me full read access to the database, including to serialized encryption keys.

What the heck?!

Well, observe that due to the magic of string interpolation I can cause the string including the Javascript to evaluate to virtually anything I want. For example, I could inject a carefully constructed Javascript string to cause the first regular expression to terminate without any results, then execute arbitrary code, then comment out the rest of the Javascript.

We can get one bit of data about any particular person out of this function: whether they are in the result set or not. However, since we can construct the result set at will, we can make that a **very significant bit** indeed. One thing Javascript can do is take a string and convert it to a number. I'll elide the code for this because it is boring, but it is fairly straightforward. After I have that Javascript, I can run a binary search to get someone's `serialized_key`. "Return Patrick if his serialized key is more than 2^{512} . OK, he isn't in the result set? Alright, return Patrick if his key is more than 2^{256} . He is in the result set? Return him if his key is more than $2^{256} + 2^{255}$"

If their key has 1,024 bits (wow, so secure), it will take me roughly 1,024 and change accesses to find it. That will take me, hmm, a minute? Two? I can now read your messages at will.

I think MongoDB will let me do all sorts of nastiness here aside from just reading parts of the person object: for example, I strongly suspect that I can execute state-changing Javascript (though I didn't

have any luck making a Lil Bobby Tables to drop the database, but I only spent about two minutes on it) or join the Person document with other documents to read out anything I want from the database, such as User password hashes. That might be a fun project for someone who is not a complete amateur.

Code injection: **fun stuff** for attackers, not quite so fun.

How to avoid this:

Don't interpolate strings in queries sent to your database! Use the MongoDB equivalent of prepared statements. If MongoDB doesn't have prepared statements, **don't use it for your security-critical projects** until it does, because you *will* be exploited.

Take Care With Releasing Software To End Users

Since making my public comments, I have heard — over and over again — that none of the above matters because Diaspora is in secret squirrel double-plus alpha unreleased and early adopters know not to put any data in it. **False.** As a highly anticipated project, Diaspora was guaranteed to (and did) have publicly accessible nodes available within literally hours of the code being available.

People who set up nodes might be intelligent enough to evaluate the security consequences of running them. That is actually false, because there are public nodes available, but we'll run with it. *Even if* the node operators understand what they are doing, their users and their users' friends who are invited to join The New Secure Facebook are not capable of evaluating their security on Diaspora. They trust that, since it is on their browser and endorsed by a friend, it must be safe and secure. (This is essentially the same process by which they joined facebook — the zuckers.)

How would I have handled the Diaspora release? Well, candidly, I wouldn't have released the code in the current state, and instead would have devoted non-trivial effort to securing it prior to release. If you put a gun to my head and said "Our donations came from 6,000 people who want to see progress, give me **something** to show them", I would have released the code that they had with the registration pages elided, forcing people to only add new users via Rake tasks or the console. That preserves 100% of the ability of developers to work on the project, and for news outlets to take screenshots, without allowing technically unsophisticated people to successfully sign up to the Diaspora seed sites.

I don't know if the Diaspora community understands how bad their current security posture is right now. Looking at the public [list of public Diaspora seeds](#), while the team has put a bold disclaimer that the software is insecure (which no one will read because no one reads on the Internet — welcome to software, guys), many of the nodes are explicitly appealing as safer options which won't reset their DB, so you won't lose your work if you start on them today. That is irresponsible.

Is Diaspora Secure After The Patches?

No. The team is manifestly out of their depth with regards to web application security, and it is almost certainly impossible for them to gather the required expertise and still hit their timetable for public release in a month. You might believe in the powers of OSS to gather experts (or at least folks who have shipped a Rails app, like myself) to Diaspora's banner and ferret out all the issues. You might also believe in magic code-fixing fairies. Personally, I'd be praying for the fairies because if Diaspora is dependent on the OSS community **their users are screwed**. There are, almost certainly, exploits as severe as the above ones left in the app, and there almost certainly will be zero-day attacks by hackers who would like to make the headline news. "Facebook Competitor Diaspora Launches; All Users Data Compromised Immediately" makes for a smashing headline in the New York Times, wouldn't you say?

Include here the disclaimer that I like OSS, think the Diaspora team is really cool, and don't mean to crush their spirits when I say that their code is unprofessional and not ready to be exposed to dedicated attackers any time soon.

Posted in [Rails](#), [Ruby](#) | [117 Responses](#)

[Automatically Minimizing Javascript and CSS with Rails](#)

By [Patrick](#) on

If you obsess over your YSlow and/or Page Speed performance scores, you might be wondering how to get your Rails app to automatically crunch Javascripts and CSS files without actually requiring cumbersome rake tasks to be repeated during every deploy. Fear not: you can quickly monkeypatch Rails to do this.

1. Copy [this gist](#) into your lib/ directory.
2. Create a config/initializers/javascript_minimization.rb as described [here](#).
3. Use your javascript_include_tag and stylesheet_include_tag helpers as normal, with the caching option turned on.
4. Watch your users save 100kb.

Posted in [Uncategorized](#) | [2 Responses](#)

[Organized, Curated Lists of Best Posts From This Blog](#)

By [Patrick](#) on

When I started this blog four years ago, I never thought it would be seen by more than a few dozen people, so I never planned any sort of information architecture to it. 500 posts (300,000 words!) and several hundred thousand readers later, it is unwieldy to get to the good stuff unless you have been following along for the last couple of years, or have a week free. To make this a little less annoying, with the help of an assistant from Hacker News I found the [best 72 articles I've written](#), grouped them by category, and for ones which are logically related explained what the connections are (in particular, for the "experiment" / "results" pairs).

I hope you find it useful. If I let out anything or if there is a category you know I publish on but did not include in the list, please let me know in the comments.

Posted in [Uncategorized](#) | [8 Responses](#)

[New Trends In Startup Financing Explained For Laymen](#)

By [Patrick](#) on

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

Did that sound like Greek to you?

Great, you might benefit from this translation of Financier into Geek. (P.S. If you haven't figured out the significance of it originally being written in Financier instead of in Geek, please, think it through.)

I originally wrote it as a [comment on Hacker News](#) but somebody asked me to put it somewhere easily findable. I have elaborated with standard blog post formatting and graphs where I thought they helped

the explanation:

Why We Care About Angel Investing

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

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

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

Why Angel Investing Frustrates Founders

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

Startup founders are, understandably, frustrated by this.

What “Valuation” Means

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

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

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

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

For example, take the company from earlier. We said it was worth \$100 prior to receiving investing, but

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

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

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

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

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

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

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

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

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

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

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

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

How Does A Valuation Cap Work?

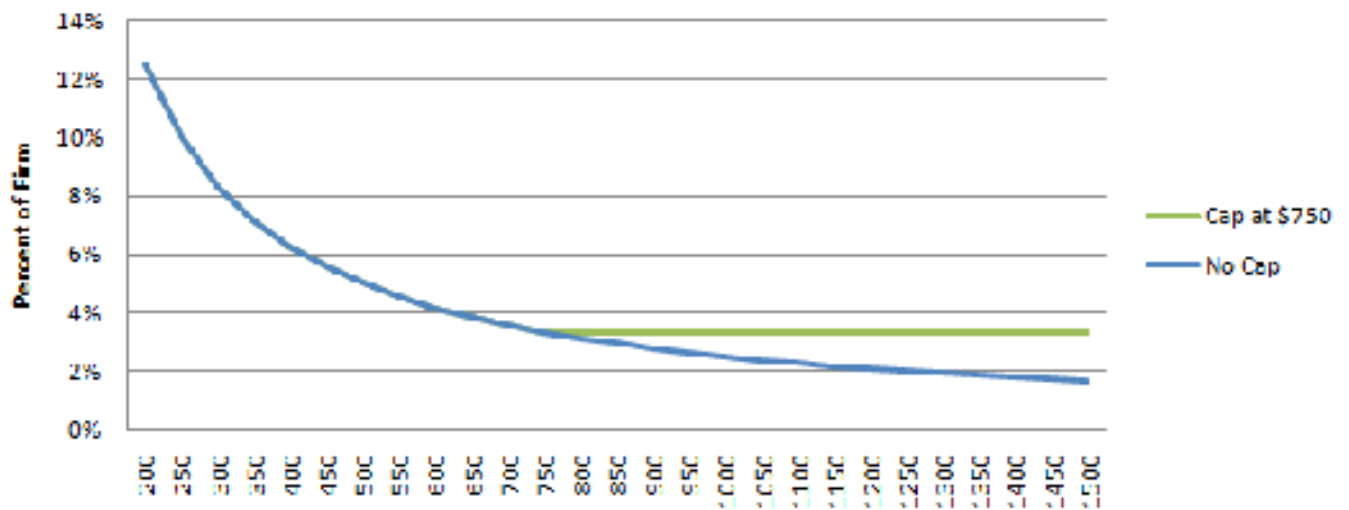
We haven't discussed valuation caps yet. Valuation caps are intended to prevent the startup dragging its feet on raising money, thus building up lots of worth in the company, and then the angel investor getting cheated. For example, if they had just grown through revenues for a year or two, they might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

Perhaps This Will Be Clearer With A Picture

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

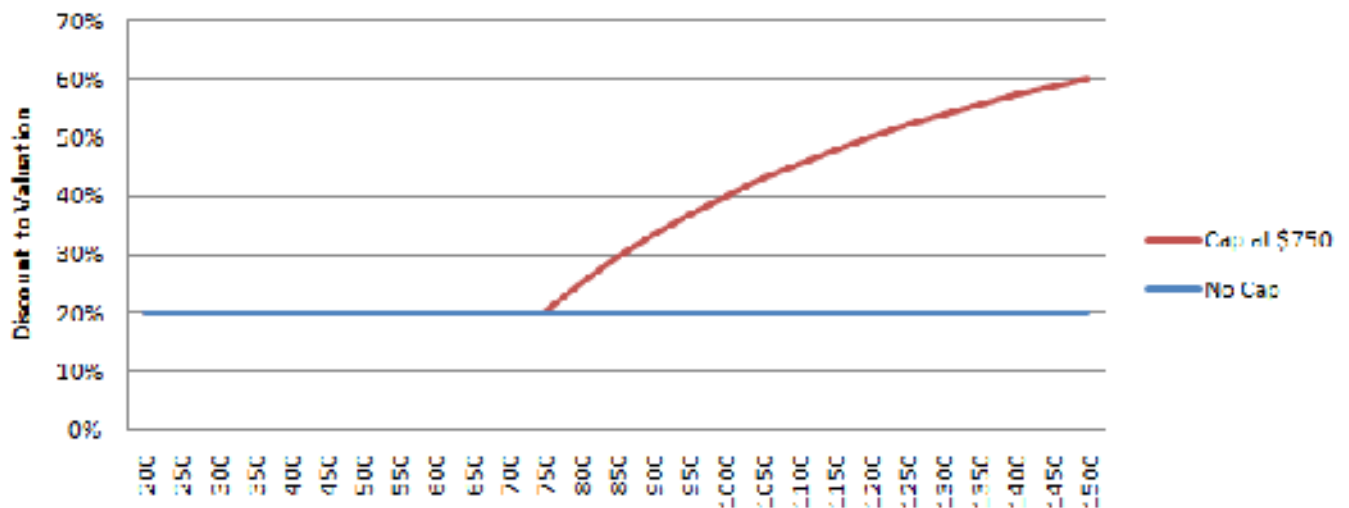
Let's see how valuation caps affect how much of the company you end up with. The better the company is doing by Series A time, the less of the company the angel ends up with. This shows the incentive for the founders: do as well as you can prior to raising money, which is the **same incentive founders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

Angel Ownership After Series A at Various Valuations



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

How Valuation Caps Affect Angel Returns (Big % Better For Angel, Worse For Founder)



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

Lest anyone get the wrong impression, my familiarity with angel investing is very limited and, to the extent that it exists, it is mostly about angel investing in small town Japan. (Oh, the stories I can't tell.)

The above explanation is based on me processing what I've read and trying to prove that I understand it by explaining it to other people. If I have made material errors, please correct me in the comments.

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

[Edit: Want to use some or all of this, including the graphs, for your own purposes? [Go ahead.](#)]

Posted in [startups](#) | [13 Responses](#)

The Hardest Adjustment To Self Employment

By [Patrick](#) on

I apologize in advance for spelling mistakes, because I am writing this on my iPad on the bullet train to Tokyo. Wonderful device, not so great for writing lengthy blog posts like my usual.

I am on the way to Tokyo because a high school friend is there this week. As soon as I heard, I told him to pick a day and I would be there. What day? Literally any day. My schedule is infinitely flexible.

That is what scares me the most about this job. Like most people, I have lived an entire lifetime conforming to schedules. They exist like the Greek gods: you didn't ask for them but they are there, there is no negotiating with them, and prolonged association means you are likely to get your dignity violated by a bovine.

But schedules are structure, and structure helps. Be at school at nine AM. Seminar starts at 10:30, do not be late. Work starts at nine, Patrick, waltzing in at ten annoys people even if you are contractually permitted to do it and even if you will still be here at 2 AM. (Regardless of start time. If you thought the gods were rational, you have been reading the wrong mythology.)

At the very least, schedules put you and everyone else on the same page as to what you should be doing. Gainfully employed young men should be *working* at 2 PM on a Wednesday. I was having a late lunch and reading a novel at the coffee shop. The waitress asked me, confused, whether I was a student or not. Students have social license to do bugger all for a few years prior to working for a living. I told her I run a software company, and one of the perks is that I get to have lunch whenever. She was impressed, but asked how my customers and employees stand that.

That's the the thing about schedules: once you have one everyone else needs to, too, preferably as close to yours as possible. My customers do not share my schedule: most use my software when I am asleep, and mail me with their urgent issues at 3 AM in the morning Japan time. I spend lots of effort decoupling their happiness from my personal availability. This does wonderful things for site uptime but it also means, perhaps regrettably, that there is generally no compulsion to work *today*.

My freelancers largely don't share my schedule either. I just got the front page for [Appointment Reminder](#) redone after several months with placeholder graphics. The extraordinarily talented designer I worked with (Melvin Ram at Volcanic Web Design, a [web design company](#)) "met" with me for a total of perhaps twenty minutes, and we worked (him on having graphical inspiration, me on wrangling it into a functioning product) in temporal isolation for the rest of the project. This is the arrangement I almost always use for freelancing. It is wonderfully productive except in that it lets me off the hook for causing forward progress.

That is the other part about scheduling: with the debatable exception of my consulting work, I am terrible about setting and keeping multi week schedules for milestones. This never came up when I was employed, since I had managers to crack the whip and avoided doing anything multi week for my business, but it is now biting me with a vengeance. I wanted to have AR in beta six weeks ago. Between consulting, vacation, and BCC, I haven't made almost any forward progress on engineering.

I know that to be true for AR because code isn't getting written, but I always think it to be true for BCC. It turns out that I am smoking something: I ran a shell script to compare my productivity (commits, A/B tests, etc) prior and post quitting. I thought it would show me spinning my wheels. Turns out I am getting more done than ever. This is normally the point where I would paste a graph but, sorry, iPad. Suffice it to say I have run more A/B tests this summer than in the last year. (Interesting finding of today: Google Checkout really does increase conversion rates over having only Paypal as an option. I strongly suspected that, but now I know.)

Sales are up, too. Why doesn't it *feel* this way? I think after a couple of decades of living by the clock I have become habituated to measuring my productivity that way. Insane and irrational, I know.

I am looking for ways to hack this: the discipline and social validation of having a schedule, without actually having to work at nine. I have been considering getting an office just to have mental separation between work and non work. (They give them away in this town if you work in tech. I could pay the rent with the savings in my iced cocoa budget.) Plus, if I have an office, I have an offsetting factor the next time I am accused of being unemployed. Sounds funny until it happens from a police officer who does not quite understand immigration statuses. (You know that controversial immigration policy in Arizona? Don't ask me my opinions about it around sharp implements. Suffice it to say I can vividly imagine what getting stopped under it will feel like.)

Another way is, and you might laugh, a little iPad app called EpicWin which gives me fake RPG loot for making progress. Will it work? No clue, but one week in, I seem to be getting more of my "boring" chores accomplished. (I had considered building this into my business for a while, but resisted because I thought it would cause neglect of my nonbusiness priorities. It turns out that, if anything, I have the opposite problem now that I have infinite scheduling flexibility.)

And I just earned 100xp for this blog post.

Posted in [microISV](#) | [17 Responses](#)

[Next Page »](#)

Teachers! Our software for printing bingo cards and resources for teaching are located at our other site. Click the image below to go there.



Small business owners! Our appointment reminder software is located at our other site. Click the image below to go there.



This blog has a [podcast](#) through [HearABlog \(RSS feed\)](#).

People are Googling for patio11 blog due to the amount of time I spend posting under that name.

About Me

- [Greatest Hits](#)
- [Start Here If You're New](#)
- [About This Blog](#)

Recent Posts

- [Getting A New Product Off The Ground: Part Two](#)
- [Getting A New Product Off The Ground: Part One](#)
- [How A Half-Broken Halloween Promotion Smashed Revenue Records](#)
- [How To Use SSL To Secure Your Rails App Against FireSheep And Other Evils](#)
- [Lessons Learned At Business of Software 2010](#)

Recent Comments

- [shapewear](#) on [Strategic SEO for Startups](#)
- Simon Brown on [Getting A New Product Off The Ground: Part Two](#)
- Robert on [Falsehoods Programmers Believe About Names](#)
- phpinfo() on [Falsehoods Programmers Believe About Names](#)
- [Want to Sell Online? First Decide WHAT You Want to Sell - Taking the "dis" out of disABILITY - Enabled4Success](#) on [Getting A New Product Off The Ground: Part Two](#)

Categories

- [30days](#)
- [ab-testing](#)
- [advertising](#)
- [AdWords](#)
- [Analytics](#)
- [Appointment Reminder](#)
- [bingo](#)
- [blogging](#)
- [bugs](#)
- [Checkout](#)
- [customer service](#)
- [Japanese](#)
- [Kalzumeus](#)
- [marketing](#)
- [microISV](#)
- [Paypal](#)
- [piracy](#)
- [popular](#)
- [Rails](#)
- [Ruby](#)
- [search](#)

- [SEO](#)
- [sight words](#)
- [startups](#)
- [stats](#)
- [support](#)
- [taxes](#)
- [teaching](#)
- [Uncategorized](#)
- [web design](#)
- [Yahoo](#)
- [YSM](#)

Copyright © 2010 [MicroISV on a Shoestring](#).

Powered by [WordPress](#) and [Hybrid](#).

☺