START

MARKETING

The Day You Start Coding

(and other essays)

How to market your startup, live the entrepreneurial mindset, manage your company, retain developers, and become a Micropreneur

- ☑ Why you should start marketing the day you start coding
- ☑ Why focusing on traffic can kill your startup
- ☑ Three words that increased my sales 1000% overnight
- ☑ Why a link from TechCrunch will not make you rich
- ☑ Why free plans don't work
- ☑ Why startup founders should stop reading business books
- ☑ The inside story of a small software acquisition

ROB WALLING

WHAT YOU CAN DO WITH THIS BOOK

You are given the unlimited right to print this book and distribute it electronically (via email, your website, or any other means). You can print out pages and put them in your office window or your taxidermist's waiting room. You can scrawl the author's words onto the sidewalk (in easily-removable chalk, of course), or hand out copies to people you meet on the street. You may not alter this book in any way, though, and you may not charge for it.

DOWNLOAD IT

This book is available from *http://www.softwarebyrob.com*

ABOUT THE AUTHOR

Rob Walling has been starting companies for most of his life and is author of the book *Start Small, Stay Small: A Developer's Guide to Launching a Startup*.

His blog, *Software By Rob*, is a top 20 startup blog and is read by tens of thousands of startup founders each month. It covers bootstrapping, software entrepreneurship, and startup marketing.

Rob owns the leading ASP.NET invoicing software on the market in addition to a handful of profitable web properties. You can reach him at at rob@softwarebyrob.com.

ABOUT THIS BOOK

This book is free. I encourage you to email, forward and otherwise share it with the world.

© 2011 Rob Walling

Design & Layout by Rajasekar (*designstudio.raj@gmail.com*)

COPYRIGHT INFO

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs License.

To view a copy of this license, visit *http://creativecommons.org/licenses/by-nc-nd/2.0*.

Contents

Introduction

I started Software by Rob in June of 2005 with a handful of bad posts and the uninteresting mission of "explain[ing] software development in terms that even the people in finance can understand."

I'm pleased to say that mission fell by the wayside after the first few days.

Over the past 6 years I've published 383 posts, received over 2000 comments (about 10x that in spam), and received just under 900,000 visits from 211 countries/territories (including several that no longer exist).

Looking through my archives a few months ago I realized that many posts are as relevant today as they were the day they were written (whether that was three days, three months, or three years ago).

The goal of this collection is to breathe new life into articles that have fallen into the archives, and to provide you with a portable, well-formatted, and easy-to-read version of the best content from the first five years of Software by Rob.

My hope is that have the same impact today as the day they were published. And trust me… there's a lot more where this came from.

Rob Walling

March 10, 2011

www.SoftwareByRob.com

P.S. If you enjoy this eBook, please let me know on Twitter (this helps me know if I should do things like this again in the future):

@robwalling Enjoying your free 'Best of Software by Rob' eBook: *http://bit.ly/hY0p6y*

Startup Marketing

Losing People Through the Bottom of Your Funnel

I had a realization recently while talking with a *Micropreneur Academy* member: focusing on increasing traffic and improving conversion rates is a fantastic game plan for a startup. But if you offer a recurring service a third step is required: retaining your existing customers.

If you're focusing hard on optimizing your *website sales funnel* it's easy to ignore what existing customers are saying. It's also hard to prioritize what to work on next: traffic, conversions or new product features. And traffic and conversion rates are the forces that grow your business.

But adding new features is at times more important than focusing on sales. I talk a lot about marketing/sales on this blog because it's crucial to your success, but in this case there's a real trade off between growing revenue and keeping your current customers happy.

So to put this idea into perspective I've started referring to a customer cancellation *as losing someone through the bottom of your sales funnel*.

The Third Leg

With sales funnels, the key metrics that you control are your traffic (how many people you send into the top of your funnel) and you conversion rate (how many people make it to the next step in the funnel).

You send traffic through SEO, AdWords, blogging, podcasting, tweeting, social media and other methods you've no doubt heard about. Since traffic is so hip *everyone's talking about it*.

You improve a conversion rate by improving that step in the process, typically using A/B testing (I recommend *Google's Website Optimizer* for website A/B testing, but any A/B testing engine will do).

But if you own a recurring product, once you've gone through the effort of convincing someone to buy, it's painful if they cancel after a single month.

The Bottom of Your Funnel

For the sake of this example let's say you have a Software-as-a-Service (SaaS) application.

Since you did your niche research and built a mailing list, your app launched with a bang and you received several sign-ups in the first week. Since you had some high quality traffic sources to your conversion rate was high, and you will never have an easier time of driving traffic to an application as when it launches. So you're set for the first two legs of this race.

But odds are you're going to begin receiving feature requests from customers almost immediately. Since you launched with the minimal feature set possible in order to begin generating revenue and get real feedback, this is expected and it's a great situation to be in.

However, you can easily fumble the ball at this point. There is a critical time in the first 1-2 months after launch where you need to crank through the initial feature requests or you will begin to lose customers through the bottom of your funnel.

You will drive traffic and convert that traffic to customers only to have them cancel because they aren't happy with your service.

Letting this happen is worse than not improving conversion rates. Every customer who cancels due to a lack of features (or any other fixable reason) translates to time and effort you spent sending them to your website and converting them into a customer. Think about it this way: if your conversion rate is 1%, losing a single customer is like 100 people never coming to your website. Strike that traffic from your logs.

I'm a bit weepy just thinking about it.

The Solution

The solution is first to realize this is happening, and second, to stop the leak before working on anything else. Adding the handful of key features that will retain most of your customers is higher priority than improving your conversion rate, and definitely higher priority than generating more traffic.

Thinking of customer cancellations as the third leg of your sales funnel helps you realize that the health of your business depends on all of these activities, and when any one is "leaky" you are losing customers one way or another.

If you can't get any customers into your funnel you will never make a sale.

If you can't convince some percentage of prospects to move to the next step in your funnel, you won't sell any product.

And if you can't stop customers from flowing out through the bottom of your funnel, you will die by attrition.

You work too hard generating traffic and converting it to customers. Don't waste it. Plug your funnel.

Original Location

http://www.softwarebyrob.com/2009/12/15/startup-marketing-mistake-losing-people-through-the-bottom-of-your-funnel/

Why Focusing on Traffic Can Kill Your Startup

Here's an interesting exercise: find a startup or microISV founder and ask the following:

What are the top 3 approaches you use to find customers?

The most common responses will involve search engine optimization, AdWords, blogging, podcasting, and perhaps social media. And this is good – these methods can drive substantial traffic to your website.

Then ask about the next step in the process:

Once you have traffic coming to your site, how will you turn prospects into customers?

This is where you'll be greeted by open-eyed stares, head tilts and puzzled smiles. Most startup founders think about driving traffic to their website. Almost no one thinks about improving conversion rates. *Why is this?*

Good Marketing Looks Easy

The most likely reason is that good marketing, as with good design, looks easy. So easy it's virtually invisible.

When you are smitten by a marketing message enough to plunk down your hard-earned coin it's almost certain that you have no idea how they convinced you to pull out your credit card. If you did, you wouldn't have pulled it out in the first place.

No, good marketing doesn't typically look like marketing. I've been reading Joel on Software for eight years and I've been a paying user of FogBugz for nearly four. But I've never felt marketed to because Joel's marketing is the best kind: well executed. So well executed it's darn near invisible to the naked eye.

What this means is that most of us think that good marketing is easy. Just like when you look at an incredible website design it seems so simple – totally stripped down. You often think "I could do that – it's just some text in a particular font and a plus sign." Only, it's not.

The designer likely started with a more complex design and whittled his way down for hours to arrive at the simple look. A look that was actually very difficult to create.

To create your own highly converting website you have to realize the effort and the process behind putting one together and not assume that putting up a website is going to automatically convince people to buy your product.

Popularity

Another reason people focus on traffic instead of conversions is because that's what everyone talks about. For every 50 books or blogs on search engine optimization or AdWords there is one on improving conversions.

And with that much discussion on the subjects of driving traffic it's no wonder people naturally put a lot of importance on it.

Traffic is Fun. Conversion, Not So Much.

Traffic is actually fun to generate. I realize this sounds sick and wrong, but getting good at generating traffic is not only enjoyable, but instantly rewarding.

Traffic is reasonably easy to generate and measure. Using tactics described in the 50 books/blogs on any given traffic generating strategy you can move hundreds of visitors to your website in no time. And you can witness the fruits of your labor rather quickly through your friendly neighborhood analytics package. It's instant gratification.

But conversions suck. It takes forever to run an A/B test, and even longer to run enough A/B tests to substantially improve conversions. Improving conversions requires patience, long-term thinking, and an attention span longer than your typical YouTube video. Something most people don't have.

As an aside, if you don't have experience improving conversion rates using A/B testing, I recommend Google's Website Optimizer. I recorded a screencast that shows you how to setup an A/B test in under 4 minutes *here*. For more info on A/B testing check out *this link*.

So What's the Answer?

As I've launched and revamped websites the process has become almost formulaic.

Every website starts with no traffic. Then you do some marketing and you get traffic, but your conversions are terrible. You never nail conversions from the start. If you can do this dinner is on me when you come to Fresno.

So you have some traffic and few sales, what do you do next? Most founders think: drive more traffic! Wrong.

The right answer is to work on improving your conversion rates before spending more time and money driving traffic.

In un-optimized sales websites I typically see conversions in the 0.1% to 0.5% range. You have to drive a lot of traffic to make a profit with conversion rates that low. Compare that to optimized sites, which should have conversion rates between 0.7% and 4% (depending on the price point and quality of traffic).

So you have a choice: you can double traffic, or double your conversion rate. I've been doing this a long time, and I assure you that doubling your conversion rate is far cheaper and will yield many more sales in the long-run than doubling traffic. Especially with a completely un-optimized sales website, which is the easiest kind to improve.

But almost no one takes this approach.

In fact, I take it a step further. I typically stop all active traffic generating activities on websites that are not converting well until I can improve conversion rates through A/B testing.

I find the typical traffic/conversion process goes as follows:

- Build some traffic. A few hundred visitors per month, maybe a thousand.
- Improve the conversion rate.
- Send more traffic.

- Improve the conversion rate some more.
- Send more traffic.
- Retire to the Bahamas

As you drive more and more traffic to a website over its lifetime this can mean the difference between generating enough money each month for a car payment…and a house payment.

NOTE: *I haven't made it to step 6 yet. But I have seen conversions increase up to 10x through A/B testing.*

Original Location

http://www.softwarebyrob.com/2009/12/09/why-focusing-on-traffic-can-kill-your-startup/

The Nine Levels of Traffic Quality

By now we've discussed the fact that you should first *plug your funnel*, then *improve conversion rates*, then work on sending as much traffic as possible to your website.

But we haven't talked about how big of a role traffic *quality* plays in determining your conversion rate.

Traffic Quality

By "quality" I mean the following: how close each visitor is to your ideal customer, and how much of a relationship you have with that visitor.

High quality traffic means each visitor is very close to your ideal customer and they know and trust you.

By the way, this is why TechCrunch traffic *is not profitable for startups*. It's completely un-targeted (unless your niche market is other startups) and you have no relationship with that audience. Using our definition above, this traffic is of very low quality.

This becomes apparent the first time you send an email to a list that you've been communicating with for some length of time. Your conversion rate for these leads will be astronomically higher than your standard website traffic, as much as 10x higher and in the worst case 2-3x higher. This is because the quality is so much better.

I regularly see a 200% differences in *Academy* sign-up conversions based on the source (and thereby the quality) of the traffic.

The reason this is important is because you have to understand where to focus your energy when driving traffic. If people from your email list will convert at 5-10x the rate of someone finding your site through Google, you can spend 5-10x the effort getting people on your mailing list and still have a break-even ROI.

Likewise, if you see the massive amount of traffic coming from SEO you have to know how many of those people are buying your product. Without that knowledge you are flying blind and cannot properly allocate your efforts.

Luckily, Google Analytics can spell this out for you using goals. For more info on setting up goals check out *this link*. You will not regret doing this. You will instantly be able to see that some sources of traffic do not convert *at all*. And you can stop pursuing that traffic and focus your efforts on methods that convert.

As an example, *Bidsketch* is a product launched by a *Micropreneur Academy* member. A few weeks ago I was writing up a detailed case study of its launch (published inside the Academy).

As I was looking at traffic stats I noticed a huge increase in mid-November due to a number of write-ups on startup and web-2.0 blogs. But the number of conversions stayed about the same as it had been the week before, meaning the conversion rate (the number of sales per visitor) plummeted.

Is this bad?

No, as long as you know *why* this is happening. And the reason, of course, is traffic quality.

Removing the Junk

In fact, my next step was to look at all visitors that stayed longer than 5 seconds, and that removed 67% of the traffic. In other words, two-thirds of the traffic from the startup and web-2.0 blogs stayed less than five seconds on the site. It was obvious they were there for a quick peek. Including them in any kind of conversion rate calculation would be a mistake. And thinking you had a major win by being listed on these blogs would also be a mistake.

Sure, it's nice to have several thousand people see your website, but if they don't convert they may as well have not shown up at all. And don't think you're building a brand – you're not Coca-Cola. The minute those users leave your site you are out of their mind forever.

General Rules

It's impossible to say unequivocally which traffic is better for every website, but having launched or revamped over 20 revenue-generating websites I've noticed a definite pattern in traffic quality based on the source.

Here is my list, in order of quality:

- Mailing list (assuming you have a relationship)
- Your blog (assuming you have a relationship)
- A referral *link from a targeted website* with a positive write-up about your product
- Direct traffic (this typically means someone heard about your product on a podcast, read about it in print or in an online article with no link, or the person is a repeat visitor that remembers your URL)
- An organic search on your product name

- A referral link with no write-up or from a non-targeted website (such as TechCrunch)
- All other organic searches (assuming this is their first visit)
- Google AdWords
- Banner and other advertising

I can point to exceptions to the order of the items above, but in general the trending follows this list.

The Moral

Blog to build your RSS and email subscriber base for the highest quality traffic. Participate in blog and podcast interviews for the #3 and #4 spots. You should always perform on-page SEO since it's a simple step to take, but only move down the list above if you have exhausted the first several traffic levels (and you will not do this until you've achieved a decent level of success).

Focus your time on high quality traffic and your high quality traffic will focus its time on you.

Original Location

http://www.softwarebyrob.com/2010/01/12/startup-marketing-mistake-ignoring-traffic-quality/

Three Words that Increased My E-commerce Sales 1000% Overnight

Until a few weeks ago I owned one of the top ranking sites on Google for the search term "beach towels." This meant I received around 2,000 visitors each month in the fall and winter, and up to 5,000 per month during spring and summer.

The problem was that when I'd purchased the site 18 months ago the conversion rate (the rate at which it converted visitors to buyers) was hovering right around 0%.

Correction…it was 0%.

Always Be Testing

The biggest lesson I learned while taking the site from a few hundred to a few thousand dollars a month (revenue, not profit), is that there are a lot of great theories about marketing, but no one can tell you the exact thing that's going to lead to an uptick in sales.

With the goal of increasing the conversion rate I read piles of books on *internet marketing*, *web analytics*, and *copywriting*. Of course, actually *doing something* lead to the real leaps in my understanding of how to help people move from browsers to buyers.

And by far the biggest lesson I learned is that *you* have to test *everything*. You use your experience and rules of thumb to come up with ideas to try, and then you have to try them and test to see if they work.

Of course, this is exactly what I didn't want to hear.

As a software developer I want things in neat little boxes. I want solid answers. Just as I go to my friend who's a brilliant SQL developer to help me with my HAVING clause, I want to go to a marketing genius and have him/her tell me the exact steps I need to take to begin converting visitors to paying customers.

But the problem is that marketing isn't like coding. Coding is a highly constrained environment and, with most problems, a well-known path to success.

Marketing is different – the options are infinite and the paths to success are unique to each problem.

Marketing Isn't Coding. Marketing is Design.

NOTE: *by "Design" I mean technical (application) design.*

Design is a less constrained environment than coding. Design is a blank sheet of paper with no syntax highlighting, no compiler, a few rules of thumb, and a lot of experience.

If someone asks you to write a function that generates a random string you probably have a pretty clear picture of the code you're going to write. You don't have to consider the possibility that the compiler thinks orange towels are out of fashion or that it's trying to save money because there's a recession.

But if someone asks you to design a *performance management* system, there are a lot fewer constraints that you have to work with, which is simultaneously a blessing and a curse (for most developers, too many options is a curse).

The specifics of your design will be heavily influenced by your past experience designing applications, and by the human factors that come into play when designing anything that interacts with people. In design, trial and error (a.k.a. experience) is worth orders of magnitude more than what you can learn from books.

Such it is with marketing.

Wash. Rinse. Repeat.

As I've considered this analogy, the main difference I've found between online marketing and design is the speed of the test cycle.

Today you might design an application that hits production in 6 months. At that point the rubber really meets the road and you find out if your design is performant, scalable, and maintainable.

With enough visitors, your internet marketing test cycle can be as short as a few days.

With such a short test cycle, it's easy to try some pretty crazy things since you can undo them in a matter of minutes. With this in mind I set out testing a stack of crazy ideas on my beach towels site.

I did this for about 6 months; I updated the graphic design twice, tried two different shopping carts, added new products, changed category names, added a personal greeting on the home page, adjusted shipping costs, added an 800 number, and made about 20 other changes.

Each time I made a change I waited for a few days to see if there was a noticeable difference in sales. But none of them made a difference. Until I added three small words…

"Low Price Guarantee"

Aargh.

What kills me is that being the low-cost provider is bad. Unless you're disruptively low-priced (like Southwest Airlines and Wal*mart), being the low-cost provider is a recipe for price wars, commoditization of your offering, and a sign that your marketing department is not very creative. I have never entered a market (including this one) with a plan to be the cheapest.

So adding this phrase wasn't on my radar for months. In fact, I made the change on a whim one afternoon and forgot about it until sales started pouring in the following day. This single change sent sales from $210/month to $2200/month immediately.

The lesson here is to be the cheapest provider in any market and you will multiply your sales by tenfold. No, wait! The *underlying* lesson is to make your customers feel at ease with what they are buying. And to do this you have to know your customer.

People buying beach towels from a website are doing it because they want to save time. They want to find a towel and make the purchase as quickly as possible. They want to feel good that they are making the right decision about their purchase, which is what "Low Price Guarantee" offers.

It gives them permission to buy here and stop surfing around looking for the best deal because they've found it.

It offers the promise that they don't have to continue down the list of Google results. If they can find a towel they like, they can check this task off their list. No one goes online to window shop for beach towels; people want to get in and get out while still feeling good about their purchase.

So the real moral is three-fold:

- Know your customer.
- Make your customer feel at ease with what he/she is buying.
- Always be testing.

And the honorary 4th:

Never use a compiler that thinks orange towels are out of fashion. Everyone knows that orange is the new pink.

Original Location

http://www.softwarebyrob.com/2009/02/10/marketing-is-design-three-words-that-increased-my-e-commerce-sales-by-1000-overnight/

Why "Luck" is a Terrible Marketing Plan for Your Startup

I've heard for years about the importance of finding a market before building a software product.

"That's ridiculous!" I would think, "How can you find a market for something before you build it?"

Years later I've realized that the single most important factor to a product's success is not the founders, not the marketing effort, and certainly not the software itself.

Nope. It's whether there's a group of people willing to pay for it.

I've developed a saying. It's short and witty and it even kind of rhymes:

Market comes first, marketing second, aesthetic third, and functionality a distant fourth

Actually, it's long, dull and kind of boring…but it's true. Oh, so very true.

Market Comes First…

The product with a sizable market and low competition wins even with bad marketing, a bad aesthetic, and poor functionality. Think QuickBooks, Palm, IE5.5 or any niche product you've ever seen that looked like it was written by a six year old but sells hundreds of thousands of copies.

You can sell garbage to a hungry market and make money. Of course, someone will eventually come and eat your lunch if you don't improve your product (the three examples I listed are in a world of hurt these days). But that's an entirely different blog post.

Luck is the Exception

Something occurred to me the other day as I was mulling through this statement – it doesn't take luck into account. Startups like *Hot or Not*,*Plenty of Fish*, Facebook apps where people throw sheep, and Twitter apps where people throws @shp.

Startups where no one really understands why they became so popular; they just caught on. Like a pet rock or a hula hoop.

In these cases, the market doesn't matter because luck trumps everything. With luck on your side you don't need money, good marketing or a solid product. You just need to be lucky.

You either luck out that your idea goes viral (which happens to maybe 1 out of 10,000 startups?) or that your idea is so brilliantly conceived and executed that people clamor to find their wallets because you've solved their problem so well.

Either way it's a complete crap shoot – you don't know your application is going to be wildly successful until it happens rather unexpectedly. The guy down the street with the same skill invests the same amount of time as you have and he has 1 million users after 3 months; you have 12, one of which is your sister.

Why would someone roll this million-sided die when there are approaches with a much higher success rate that provide nearly all of the same benefits?

Because high-growth startups are where the cool kids hang out. We dream of being the next startup poster boy or girl that gets mentioned on *TechCrunch*.

You won't wind up on the cover of *Fast Company* for writing a waste management application. But you can build a very lucrative income if people in the waste management industry need your app.

The Startup Lottery

I need to stop here and say this: with a "hot idea" startup the odds are overwhelmingly against ever getting funding, much less having a liquidity event that puts money in your pocket. Your reward for success needs to be enormous (enough to never have to work again) because your odds of success are probably 1 in 10,000 if not worse.

If you accept that and realize that the 80 hour weeks for a year (or three) at sub-market wages are worth the gamble, then go for it. There is a ton of excitement and fun along the way, and a genuine passion and pride in building a startup like this…but it's a bad decision like buying a lottery ticket. The odds are very much against you, worse than any bet you can place at the race track.

But there are still those who go for the VC-funded "hot idea" startup. Aside from the cool factor why might we pursue it?

I know a handful of people – including myself – who have reached for the startup life. Each of us did it for one, *maybe* two startups (you can only do so many before you burn out). Not surprisingly, none of us "made it."

Based on conversations with these friends the reasons given for pursuing this path are the same ones I gave back when I started pursuing this road with a couple Yale MBAs. They fall into two categories:

- The "lottery" factor of hitting it big and cashing out
- Benefits you get from owning your own business (passion for what you're building, being in control, excitement, etc…)

The first reason implies that you're going to do something with the money you make. My guess is you would either plan to stop working and retire to Tahiti, or continue working only on projects you enjoy.

But why not do one of those right now without working like a dog for three years with the odds completely stacked against you? Why not take a route that instead of having a 1 in 10,000 change of success has maybe a 1 in 100 chance of success and can bring you to the same ends; provide you with a life where you could take off when you wanted or work on projects you enjoyed?

There are many ways to get there that don't involve VC funding:*Micropreneurship* is the route I've chosen, Joel Spolsky's done well with a traditional software company and 37signals has done well with a non-traditional software company.

Even someone you may not have heard of like *Stephane Grenier* makes a good living with a real software product that fills a market's need. He's probably not going to sell out for 10 million dollars, but he's doing all right and has all the benefits you'd expect from owning his own business: passion, flexibility, freedom, etc…

The common thread to the examples I've named above involve one thing: finding a group of people willing to pay for your software.

Back to Basics

Fast Company would have you believe that the "million users in 3 month" scenario is the best way to build a startup (because it makes a good story and sells magazines).

But the way the vast majority (dare I say 99.5%?) of all businesses in this world that succeed in the long-term – be they large or small, high-growth startup or lifestyle business – is to find a market that is willing to pay them money for *something*.

That *something* can be dry cleaning services, *invoicing software*, *hosted salesforce automation*, or a blueprint for *how to launch a software product on your own*… what matters is finding a group of people who need your*something* more than they need the money you're charging for it.

How do you find those people? I'll be talking about that in a future post.

But once you find them, provide your product with no hassles at a price where you make a healthy profit and you're set.

Do that, and your marketing plan won't need luck.

Original Location

http://www.softwarebyrob.com/2009/09/22/why-luck-is-a-terrible-marketing-plan-for-your-startup/

Why A Link from TechCrunch Will Not Make You Rich

When I talk to people who are thinking about launching a software product, whether a high-growth or *Micropreneur*endeavor, from time to time I hear that if they could get to the front page of Digg or get a mention on TechCrunch that they would be "set."

The problem is, your market is most likely not the people who read Digg. Nor the people who read TechCrunch.

And if it is, you're in for a tough ride. This audience is fickle, moves quickly, and looks at a lot of sites for about five seconds before clicking the back button. If you do get the big swarm of traffic run the stats on how many visitors stay longer than 5 seconds… seriously.

When I've been on the front page of Digg more than 90% of that traffic has stayed on my site for *less than 5 seconds*. That's not a market, that's a drive by.

When you receive 50,000 visitors from one of the major media sites you will be lucky to convert five sales. Five measly sales. That has to win for the worst conversion rate ever.

The reason? *They are not your market*.

When looking at your marking plan you should actually be thinking:

If I could get on the front page of [small-but-very-focused-niche-website].com…

Find the website(s) where your real market – the people who will actually buy your product – hang out. The competition will be less and your conversion rates will be orders of magnitude higher.

Original Location

http://www.softwarebyrob.com/2009/09/23/why-a-link-from-techcrunch-is-a-terrible-marketing-plan/

Why Free Plans Don't Work

The following is a guest article by Ruben Gamez of Bidsketch.

Not too long ago it seemed like every product I knew was offering some sort of free plan. The strategy was brilliant: get loads of people using your product and eventually turn them into paying customers. Everywhere I looked there were stories of people making money hand over fist with this approach.

When 37signals talked about *giving something away for free as a marketing strategy*, it made a lot of sense to me:

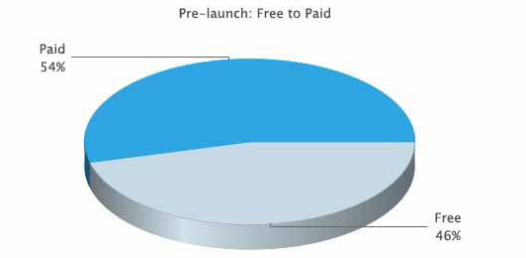
> *"For us, Writeboard and Ta-da list are completely free apps that we use to get people on the path to using our other products. Also,* ***we always offer some sort of free version of all our apps.***

> *We want people to experience the product, the interface, the usefulness of what we've built. Once they're hooked, they're much more likely to upgrade to one of the paying plans (which allow more projects or pages and gives people access to additional features like file uploading and ssl data encryption).*"

So when I launched *Bidsketch* — a SaaS based proposal application for designers — offering a free plan was a no-brainer in my book. Out of all the important decisions I spent time mulling over before my launch, I gave this one the least thought.

Early on, things were working out nicely. In the first few days of my launch I had **more people sign up for the paid plan than the free plan**.

"*Man, this free plan is really working out*," I thought. Here is a look at the numbers:



The numbers looked but great, but I suspected they weren't sustainable because I had launched to my mailing list. A well-maintained mailing list*tends to convert much better than traffic from other sources*.

In any case, I was still happy with the results a week later, once I started converting general website traffic:

While the numbers looked good I knew they wouldn't last because I was relying on a limited time offer. I just didn't realize how much worse things would get:

For the next month only 1% of users would choose the paid option. My user base was growing fast but the money was barely trickling in. Also, support was starting to get tricky, which left me uncomfortable at the thought of what things would look like six months down the line.

How many of the free accounts was I able to upgrade to paid? I didn't fare any better upselling users: 0.8% of free user accounts eventually upgraded to paid.

When things started going south, I figured I was to blame for this. I simply wasn't carving out the right features. Or maybe I wasn't prompting for upgrades at the right places.

I tried all sorts of tactics to convert my free users:

- More upgrade prompts
- Less features on free accounts
- Premium features for 15 days
- More emails aimed at upselling users to paid

None of these changes had a significant impact. The only thing that seemed to be consistent about my growth was that my revenue was relatively flat while my user base kept growing.

If I stayed on this path, I'd soon have **thousands of free users to support**.

So in a desperate attempt to get things moving in the right direction, I experimented for a week by killing my free plan. I didn't tell anyone that I was getting rid of my free plan, I simply deleted it from my pricing page.

My major concern was that I'd keep the same number of paid users coming in and I'd lose all the free ones. Which means I wouldn't have a targeted list of users to try to upsell to a paid plan. Not that I was having much success getting them to upgrade, but at least it was something.

Things didn't quite turn out that way. This change that took all of five minutes to make, led to an **8x increase in paid conversions**.

Look at that again. That's not 8%. That's 800%.

I felt comfortable enough with the results to try it out for the entire month. Amazingly, this resulted in a 10x increase in paid conversions for the month.

And I'm not the only one

It wasn't long after I got rid of my free plan that I started to notice that a lot of people were citing similar issues with having a free plan.

I saw that 37signals had hidden theirs.

Before:

	OUR BEST PLAN	Our most popular business plans			Other plans	
	Max $149/month Sign up	**Premium** $99/month Sign up	**Plus** $49/month Sign up	**Basic** $24/month Sign up	**Personal** $12/month Sign up	**Free** Free Sign up
Projects you can manage at once	**Unlimited**	**100**	**35**	**15**	**3**	**1**
Storage for file sharing	**50 GB**	**10 GB**	**3 GB**	**500 MB**	**250 MB**	–
30-day free trial	✓	✓	✓	✓	✓	free
Unlimited people & clients	✓	✓	✓	✓	✓	✓
Chat (via Campfire)	✓	✓	✓	✓	✓	✓
Secure SSL connection	✓	✓	✓	–	–	–
Time tracking	✓	✓	✓	–	–	–
Writeboards	Unlimited	Unlimited	Unlimited	Unlimited	Unlimited	2 max

After:

And thenI ran into a Mixergy interview where 37signals founder, Jason Fried, talks about their free plan (6:00 into the interview):

> *"… The majority of the revenues for our products come from people who sign up for the paid versions upfront. So we definitely have people upgrading from free to paid,* ***but the majority of people who are on pay started on pay...*** *of course, more people are going to pick the free version and stay on the free version, but if you're looking to get paying customers, ask for money upfront and you'll have a lot better shot of getting them."*

The so-called Freemium success stories had similarly low ratios of free to paid accounts. We can see *numbers published about Pandora, Evernote, and MailChimp* showing this pattern.

Pandora started out with less than 1% of their user base as paid subscribers. Once they focused on delivering a better premium offering they were able to increase that to 1.7%. Still, pretty underwhelming unless you've got **20 million** people using your service like they do.

Evernote is looking at a 0.5% conversion rate to paid accounts initially and can convert 2% of the people that stick around for a year.

While there wasn't a specific conversion rate published for MailChimp, they did mention the negative side effect of abuse-related issues:

> *"But the biggest bumps of all? A 354 percent increase in abuse-related issues like spamming, followed by a 245 percent increase in legal costs dealing people trying to game the system."*

Holy crap. Where was this info when I needed it?

CrazyEgg decided to *drop its free plan in Jan of 2009* and they haven't looked back. I asked CrazyEgg co-founder Hiten Shah why they decided to drop their free plan back then. "We thought that if we dropped it we would make more money," said Hiten. This turned out to be a good move since it **doubled their revenue that month**.

LessAccounting co-founder Allan Branch said while they don't claim to know what the best approach is in regards to a free plan, they haven't seen a good reason to change what they're doing now. With them, users have to sign up to a paid plan trial, and will get dropped to a free plan if they don't enter payment information at the end of the trial. Obviously, this approach of making users choose a paid plan at signup has worked well for them so far.

An Example We Can Relate To

A lot of us aren't at the same level that these guys are; we're not dealing with millions of users, or even hundreds of thousands. So an example like *Pluggio* might be easier to relate to.

Pluggio is a Freemium Twitter web app created by Justin Vincent. He has a great *stats page* that shows everything from monthly revenue to the breakdown of users by plan type.

Taking a look at that page reveals that he's actually doing very well for a relatively young app in this space.

He's been averaging about a thousand dollars a month since November of last year. And unlike the bigger guys, his paid users make up 2.5% of all accounts. That's damn good for any sort of Freemium app judging by the numbers that we've seen so far.

I spoke with Justin to ask him about his experience with the Freemium model. He seemed to be doing well with Pluggio which is why I was surprised when he told me he was seriously considering killing his free plan.

His reason for doing this? Revenue has been relatively flat and the number of users has been steadily increasing over the last few months (currently nearing five thousand).

This says a lot about the pitfalls of having a free plan for entrepreneurs with limited resources.

Do they ever make sense?

I'm not saying that it's impossible to be successful if you launch with a free plan.

Obviously free plans have worked well for companies like Wufoo, MailChimp, and FreshBooks, so we know they can work. But the problem is that **we're not them**.

We need to stop blindly copying them and start thinking about ways to bring in revenue.

I'll concede that there are certain types of apps that are more likely to succeed by offering a free plan and going with the Freemium model. But the vast majority of apps aren't in this category, and the vast majority of people don't have the resources to make that model work.

Taking advantage of word-of-mouth marketing requires more users than most of us will attain. Instead, we end up with a large number of free users zapping away valuable resources for nothing in return. To top it off, most free users will never end up converting to a paid plan.

If we have thousands of users that don't increase awareness and will never pay for our product, why do we insist in offering something that's going to hurt our business? Maybe we should just skip that free plan and focus on making money instead.

About the Author

Ruben Gamez is the founder of *Bidsketch*, web based proposal software for designers. When he's not developing software, he's furiously working towards becoming a better Micropreneur.

Original Location

http://www.softwarebyrob.com/2010/08/18/why-free-plans-dont-work/

Your Market is Smaller Than You Think

The subject of measuring market size comes up every few weeks in my interactions with startup founders, and one point I always raise is the difference between a top-down and bottom-up approach.

Top-Down

The top-down approach is the one you would traditionally link to business school case studies and startups going after large markets.

The approach involves going to a source of (often public) data and finding out how many X's there are in the world, where X is your target consumer…be it a barber shop, web design firm or male under the age of 34.

The challenge with this approach is that unless you have a stack of cash from here to the moon, you have zero chance of reaching all of those X's.

As a self-funded venture your best case will be to communicate with a single digit percentage of your market through online marketing and perhaps some cold calls and direct mail.

If you have funding coming out of your ears, maybe you'll drop seven figures on a Super Bowl ad and reach into a double digit percentage of your market. Of course, the effectiveness of your funnel drops exponentially when you move to mass advertising. But it worked for a lot of the dot com's in the early 2000s. Oh wait, no…that's right, it didn't.

Jokes aside, the top-down approach has its place in business plans and pitch decks commonly used to ask people for money.

Where they don't belong is with a bootstrapped founder trying to figure out how many customers she can expect to reach next month on her shoestring marketing budget. In that case, bottom-up is a better choice.

Bottom-Up

There are many bottom-up approaches to sizing a market; I discuss the approach I use in my *developer's guide to launching a startup*.

It involves using the *Google AdWords Keyword Tool* to estimate how many X's are actually looking for your solution to their problem on a monthly basis.

And while this doesn't give you an idea of the total market size, it does show you how many people you can reach right away if you nail your online marketing.

Sure, there is still a sea of prospects that can't be reached online, but for self-funded startups it's obvious that these days online marketing is a much better use of your resources than cold calling, direct mail, trade shows, etc…

I'm not going to go into detail on the approach here except to say that it uses the AdWords Keyword Tool to estimate how many prospects are searching Google for your kind of product each month. It then crunches those numbers together with an expected conversion rate and expected price to give you an idea of expected monthly revenue.

The thing is, the Google AdWords Keyword Tool is wrought with land mines. And I've realized over the past few weeks that many entrepreneurs are making the same mistakes, inadvertently bringing back results that make their market appear larger than it actually is.

This is a big deal, and one I hope to counter-act by looking at the four most common mistakes I see again and again when using the AdWords Keyword Tool.

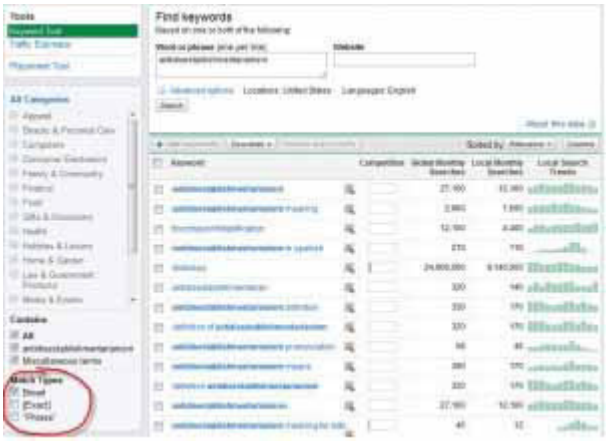
Mistake #1: Not Logging In

So you've decided to use the bottom-up approach to sizing your market. Sweet. Let's hit the *Google AdWords Keyword Tool* and find out how many people are searching for our key marketing term.

Typically you would search for a term like "inventory software," but for our new whizz-bang social media application it all depends on people searching for the longest word in the English language, *antidisestablishmentarianism*.

Don't ask me why, you need funding to understand.

So first, make sure you're logged in to your AdWords account. For some reason Google gives you limited (aka crap) results if you're not logged in, with only a tiny warning about this.



Mistake #1: Log in!

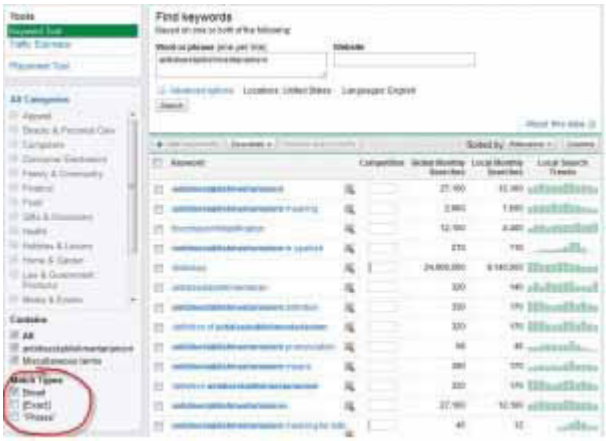
Mistake #2: Using the Default Match Type

Next, enter your term and hit “search.”

“Oh man, 27,100 people per month search for*antidisestablishmentarianism*. This is precisely our target market. I’m counting the mad stacks of cash already! “

Keyword	Global Monthly Searches	Local Monthly Searches
antidisestablishmentarianism	27,100	12,100

But hold on a sec. Did you take a peek at what kind of search you performed? Yep, way down in the lower left of the page there are three checkboxes, hidden away as if to intentionally lead you to believe your market is larger than it appears.



That's mistake #2.

You don't want *broad match*…you want *exact match*. If you are searching for the term "dog bites," a broad match will match on any of the following:

- dog bites man
- man bites dog
- someone with a hot dog bites into it

In other words, this returns way more results than you're looking for. So uncheck *broad* and *check* exact.

Mistake #3: Global Searches

Ok, not too bad. We still show 14,800 people looking for this term.

Keyword	Global Monthly Searches	Local Monthly Searches
[antidisestablishmentarianism]	14,800	6,600

But wait…those are global searches. This doesn't mean what you think it means.

"Global searches are just people searching Google from around the world, right? Well, our application is for social media so anyone in the world can use it. 14,800, here we come!"

Except not.

The real story is that the global monthly search total includes those through every Google property, such as Google.com, Google.co.uk, Google.co.in, etc…

The challenge is that Google includes some pretty localized factors when it ranks websites in these "local" search engines. Rankings include factors like having the same top level domain (so .co.in for Google.co.in) and being hosted inside that country (to name a few).

Needless to say, local websites have the advantage here, which makes sense. This provides the most accurate results for people using that search engine.

Bad news for you. Good luck ranking in every Google search engine. It's not going to happen. Cash in your chips and move on to #4.

Mistake #4: Local Searches

"Ok" you say reluctantly, "I can live with 6,600 monthly searches. Maybe we can't rely on our freemium, advertising-based, viral revenue model, though. We might have to charge someone at some point…but we can make it work."

Keyword	Global Monthly Searches	Local Monthly Searches
[antidisestablishmentarianism]	14,800	6,600

Except for one last thing.

The Google AdWord Keyword Tool doesn't show you how many visitors you would receive if you ranked #1 for this term. It seems like it should, but it doesn't.

I have a number of sites that rank #1 for a number of terms, and almost none of them match this local monthly search value. It's generally accepted that this value is higher than the number of uniques you will receive from ranking #1 for this term.

D'oh.

The rule of thumb (and it's only a rule of thumb – your mileage may vary) is to take 40 – 60% of the keyword tool's number as the actual number of unique visitors per month that you would receive if you ranked #1 in Google for this term. I tend to ballpark it at 50%.

3,300, here you come.

Original Location

http://www.softwarebyrob.com/2010/09/30/your-market-is-smaller-than-you-think/

Why You Should Start Marketing the Day You Start Coding

Photo by *DeclanTM*

Startup Marketing

This article is #7 in a series about startup marketing. The first 6 (not required before you read this one) are available here: part 1, part 2, part 3, part 4, part 5, part 6.

I've gone on and on about the subject of pre-launch marketing on *my podcast*, made mention of it in *my book*, went into detail on *TechZing*, and again on a recent *Micropreneur Academy* conference call.

And after talking about this subject at length, I found myself again evangelizing it last week at the *Business of Software* conference. That's when I realized I needed to sit down and create a permanent written resource for the topic. Then you don't have to listen to me tell you about it – you can just ask for the URL.

So the intention of this post is to lay out the key details of why you should start marketing your startup (or product, or book, or anything else you will launch) months before launch day.

This may sound obvious, but given the number of times I've been asked about it (and the number of times I've seen people do it poorly) it's apparent it needs further examination.

Objections

The two most common objections I've heard about pre-launch marketing are that:

- Someone might steal your idea
- You're too busy writing code to spend time marketing

Let me address the fear of someone stealing your idea with the following: Wake the hell up! No one cares about your idea. Not even your mom (I know she said she does, but she was just being nice).

Anyone with the skill to clone your idea and the motivation to actually make it happen is way too busy with the 37 ideas they have every day to bother taking yours. And if someone does steal it before you launch, consider it a favor.

Having your idea stolen sooner saves you the hassle of building it, only to have someone steal it then. If it's that easy to steal it's going to happen one way or the other.

Remember, *ideas are worth nothing* on their own. And these days with how easy it is to build an application, your code isn't worth much, either. This hurts me as much as anyone since I'm a developer.

But myself and another developer could get together and clone almost any popular web application in a month. Or for that matter, we could simply buy a clone script. Twitter, Facebook, eBay, Groupon, Digg, and about 50 others are available for *around $100 each*.

No, these days even technical execution is mostly trivial (with a few exceptions for apps built around unique algorithms). Far more important is marketing execution. If you can out-market someone, you can make your Git repository public and still kick the crap out of anyone.

Ideas (and in many cases the code itself) are not worth as much as we think. It's marketing that most often makes the difference between a successful and a failed startup.

Whew…now where was I?

Oh yes, the thought that someone might steal your idea is even more preposterous today than it was ten years ago.

For the other objection: you're too busy writing code to spend time marketing.

Ummm…yeah. I suppose you're also too busy to spend time compiling, using source control, or saving files to your hard drive.

If all you're doing is building a hobby project then no marketing needed. You're fine to just post it to your blog (30 uniques per month, baby!) and let it languish in obscurity.

But if you have any desire to sell your software, consider marketing a fundamental building block of the process. Without marketing, your*product* is nothing more than a *project* (something you build for fun, not money).

If you plan to sell your product, marketing is an absolute requirement. As critical to the process as saving code to your hard drive. Skip it and you're doomed.

Reasons to Start Marketing Before Your Launch

Now let's look at four reasons why you should start marketing the day you decide to move forward with your idea.

To give you a bit of background (that I'll expand upon later), the goal of pre-launch marketing is not just to build buzz, but to get permission to contact people who are interested in your product. This is best achieved by building a launch notification email list, something fairly commonly implemented these days.

We'll go into more specifics later on in this post.

Reason #1: Idea Validation

The day you decide to move forward with an idea there's a lot of uncertainty. If you've ever made the commitment to invest 400+ hours, you know how mentally taxing this can be. Especially if you've made the decision based on a hunch with little data to support your decision.

This uncertainty makes the six-month slog that much more challenging. It's hard enough to give up your nights and weekends for six months. Even harder when you're not sure anyone's going to care once you launch.

In 2-4 hours you can setup a landing page and begin collecting emails. This simple act (coupled with a small amount of marketing) can make the difference between having the confidence that you're building something people want, and having no clue if you're pouring several person-months of effort down the drain.

Don't underestimate the impact that fear and uncertainty can have on your chances of success.

Imagine yourself three months into building your product. You have three months left. You're tired because you work every night until 1am. Your wife tolerates it, but she's not happy about all the time you spend sitting in front of your computer with no money to show for it. And you haven't seen your friends in months.

It sucks.

In the above situation, assume you have 650 targeted email addresses you've compiled through some small marketing efforts and a landing page. Suddenly things don't look so bleak. You have some sales waiting for you once you push the bits to your server.

And vice versa, if you're three months in and you've received several thousand uniques to your landing page but only 6 sign-ups, you have a problem. Either your landing page stinks or your idea is a lead balloon.

Either way, you need to put coding on hold and figure out the problem.

Reason #2: Instant Beta List

I'm not a fan of open betas, but whether you're going to release your app to 5 or 500 beta testers, you have to find those people. And this is a lot harder than it sounds.

Gathering interested prospects over time allows you the flexibility to instantly email 5 people – even months before launch – and ask their opinion about a feature, design choice, or any decision better made by a potential customer than by a vote between you and your mom.

And once you're ready for get beta testing it's a slam dunk. It reduces your time to find testers from a few days to a few hours.

As an aside: unless there is a compelling reason, opt for a small beta (5 – 20 people), and offer a heavily discounted or free version to participants if they contribute opinions and bug reports.

If you decide to go with a large beta (and you'd better have a good reason for this), don't give your software away to everyone who participates. This first group of prospects is a critical source of early sales.

Reason #3: Launch Day

If you've ever launched a product without a mailing list, you know it's painful.

After hundreds of hours of development your big day arrives. You email everyone you know, flex your networking muscles, issue a press release, and end the day with three sales at $20 each.

60 bucks. Wow…how will you ever deal with such a massive influx of capital?

If you haven't started marketing, your launch day is your halfway point to having a successful product. Building it was the easy part.

Contrast that with a mailing list of 650 interested people who visited your landing page and decided your offer was compelling enough to provide their email address. You've been greasing the marketing wheels for months to get here.

You send an email letting them know you'll be launching in a week or so, then an email with a nice discount that expires after a few days. Your conversion rate should land between 5 and 40% depending on how long you've been collecting emails, the interest level of the prospects, and how compelling you make your offer.

At 5% you'll sell 32 copies. At 40% it's 260.

I assure you: selling 260 copies of your app (or garnering 260 sign-ups for your SaaS app) on launch day will do wonders for your morale.

Reason #4: Building Links Over Time

The final advantage is the ability to build links over time. Nothing fancy here – it's common knowledge that search engines look more favorably on a website with a "natural" link profile, part of which involves receiving links organically over time rather than receiving a zillion of them on a single day.

While Google won't penalize you for receiving a stack of links at launch, you will tend to rank higher for a longer duration if you gather those links over time.

Execution

Ok, I said this would be a "why" article, but I hate talking so much theory without giving actionable advice. So let's take a quick look at the details of getting setup to start pre-launch marketing. There are many variations, but here's the simplest approach:

- Buy a domain name and point it to your web host
- Setup a landing page. Keep your copy *really* short (and punchy). You need to pique interest, not convince them to buy.
- Collect emails on that landing page

Using *GoDaddy* for step 1, one of the approaches I'll mention below for step 2, and *MailChimp* for step 3, this should take no more than 2-4 hours from start to finish.

Once this is setup the major task is driving traffic, which is beyond the scope of this article (but I wrote about it in *my book*, and I'll be blogging more about it in the future).

So let's look at the best approaches for setting up a landing page:

Approach #1: WordPress with LaunchPad

This is my approach of choice. Install WordPress and install the LaunchPad theme. Edit the copy, add your subscribe form. Bam – you're done.

Elapsed time: 2 hours.

My most recent use of this approach, for *my book*, yielded a conversion rate of unique visitors to emails of just under 50%.

Here's a screen shot of the landing page:

This is all the text that appeared on the page. It's just enough to pique your interest.

Approach #2: Static HTML

I know you can hack HTML. But please don't design a landing page yourself unless you are a designer. A crappy landing page (like something I would hack myself) will have a visitor to email conversion rate around 5-10%. A well-designed page with good copy and targeted visitors should do 30-50%.

A few landing page examples from which to borrow inspiration:

- *Design Book*
- *AppIgnite*
- *55 Creative Coming Soon Pages*

Approach #3: Unbounce

I haven't used *Unbounce*, but they're a SaaS solution to this landing page issue. At $25/month for the cheapest plan it's a bit pricey for a developer who can use one of the options above for little or no cost.

And although I like their selection of *landing page layouts*, I wish they had more look-and-feel choices (they launched a few months ago so I imagine they are working on this).

With that said, Unbounce is a good choice if you're not a developer, or don't have any time to tackle one of the other options I've listed.

Your Turn

If you have any questions or war stories of pre-launch marketing please share them in the comments.

Original Location

http://www.softwarebyrob.com/2010/10/14/startup-marketing-part-6-why-you-should-start-marketing-the-day-you-start-coding/

How to Find Your 4-Second Startup Pitch

This post is an excerpt from my upcoming book, *Start Small, Stay Small: A Developer's Guide to Launching a Startup*.

If you haven't already signed up to receive the "crazy pre-release deal" when the book launches, you can do so at the *startup book*website.

One piece of your marketing that you need to nail down is your "hook."

This is not your Unique Selling Proposition, and it's not your elevator pitch. It's the headline of your home page. That single sentence that grabs the reader in and makes her know she's in the right place.

- **DotNetInvoice's** hook is "Save Time. Get Paid Faster."
- **FogBugz's** hook is "Bring Your Project into Focus"
- **Basecamp's** hook is "The Better Way to Get Projects Done"
- **Bidsketch's** hook is "Simple Proposal Software Made for Designers"
- When it was launched, the **iPod's** hook was "One Thousand Songs in Your Pocket"

These are 5-7 word summaries of your product. Each one conveys an image in your mind. Each one describes what the product does and (in most cases) who it's made for.

The benefit to finding your hook (which I also call "your 4-second pitch") is when you meet someone face to face and they ask what your startup does, it's your description. In 4-seconds you tell them clearly and concisely what it is that you do:

"We build invoicing software that helps entrepreneurs save time and get paid faster."

Finding It

To find your hook you can take one of three approaches:

- Explain what your product does and for whom. Such as "Simple proposal software made for designers."
- Make a promise to the customer espousing a benefit of your product, such as "Save Time. Get Paid Faster."
- Describe the single most remarkable feature of your product, such as "One Thousand Songs in Your Pocket."

Some other fake examples for something as boring as inventory software made for grocery stores:

- **What it does and for whom:** Inventory Tracking for Grocery Stores
- **Promise:** Automate Your Inventory
- **Feature:** A Million Items at your Fingertips

Spend a few minutes brainstorming your hook; it's surprisingly easy to create one. Keep the words short and simply; no marketing speak. Discuss it with a friend or colleague in your target market to find out if it pulls him in.

Once you've decided on a hook you should put it as the header on your home page, and consider using it as your tagline. This hook is what will allow you to tell someone in 3 seconds what your product does, or at least why it's so cool that they need to check it out.

An Invitation

Once you find your hook, post it in the comments with a link to your site. Consider this permission to do a little marketing for your startup or product.

With the purpose of this exercise in mind, keep it to one sentence plus your URL.

Original Location

http://www.softwarebyrob.com/2010/05/13/how-to-find-your-4-second-startup-pitch/

Crowdsourcing Your Product Name

Product naming is hard.

Too many factors come into play when looking for a name and it's almost impossible to decide on the right name once you've stared at the same list for a week straight. This is the kind of thing that keeps you up at night, even after you've made the decision.

And asking opinions is fine, but more often than not the people you ask are not in your demographic:

"Hi Mom. Things are good, thanks. Hey while I have you on the phone, what do you think I should call my enterprise level encryption engine?"

About two months ago Patrick Thompson, a member of the *Micropreneur Academy*, was in search of a name for his *speed reading eBook reader* for the iPhone. We emailed several times about the process he followed to find his optimal name, and given his creative approach I wanted to share it here.

As an aside, his application launched last week and you can find it at *http://www.quickreader.net/* or in the iPhone app store under the name "QuickReader."

The Approach

A while back Patrick contacted a handful of Academy members and asked for opinions on potential product names. Our task was to choose a few names we liked from a long list of ideas. From this data he culled a short list of potential names – around 40 – and put a survey on Amazon's*Mechanical Turk*. He got the idea from these two links: *here* and *here*.

He asked people to view a screencast of the application, offer suggestions for the name and give their feedback on the app. He received 29 responses and paid a total of around $7.

With a successful test run in his pocket he put out a 14-question survey looking for 500 respondents. He paid $.055 per survey (including Amazon's $.005), and had 50 respondents in the first 3 hours.

The Findings

Five days later Patrick had 250 responses and enough information to decide on a name, but let the survey run because he was receiving solid information from the other (non-naming) survey questions, as well as leads who were interested in being notified when the app launched. At the time of this writing he's at 460 responses.

While the purpose of the survey was to choose a name for his application, he (wisely) took the opportunity to also ask about demographics. Among other things, he found that:

- A slight majority of respondents were female
- Nearly 75% were between 18 and 35
- 63% were in North America; 20% in India
- 28% of respondents were in his target market (meaning they owned an iPhone or iPod touch)
- 25% of respondents had ever read a book on a mobile device (not bad!)
- More than half had an interest in speed reading
- 21% volunteered to beta test and provided their email
- 20% asked to be notified when the app was available and provided their email

Obviously the population taking the survey was skewed towards a younger, technology savvy audience. But it's close enough to his target market to work.

Regarding the product name, which was the original purpose of the survey, a list of 40 names was given to each respondent was asked to choose up to 3 names. While QuickReader wasn't the top name for the entire population, it was a close second among people that had an interest in speed reading and 3 of the top 5 overall names had the word "quick" and "read" or "reader" in them. And QuickReader works well for some of Patrick's future plans for the app.

The top name overall, throughout the survey, was iReadFast. While he decided not to use this for the product name (as it doesn't fit well for a general e-book reader), he did grab the ireadfast.com domain name to use as a blog or education site in the future. I'm surprised the name was available.

Lessons Learned

Overall, Patrick said that using Mechanical Turk for the survey was a great experience and obviously quite cost effective. It cost him $27.50 for 500 responses ($.05 per respondent + .005 per respondent for Amazon's cut). He was able to decide on an app name. He found out something about what devices Turks use and their attitudes about

reading and speed reading, and he got a list of close to 100 e-mail addresses of people wanting to be notified when the app launched.

And finally, here is the advice he provided from his research and first-hand experience:

- **Always do a small** trial or 2 of your survey task before you start the real one
- **Try to make it interesting** for the Turk
- **Providing a link to a screencast** of the app was a good idea (the Turks enjoyed it)
- **Provide a confirmation code** in your survey, on the final thank you page, once they have answered all of the questions. This is what the Turk enters in a form at the end to prove that they took the survey.
- **Executing the survey in an external survey site**, rather than doing the survey directly in Amazon's form, allows you to use the built in reports of your survey provider (rather than just getting the excel data.)
- **Keywords are important**. When you publish your task, try to add as many relevant keywords as possible.
- **Even if your goal is narrow** (e.g., selecting a name), use the opportunity to find out more about your potential users and market.
- **Always have a question** asking how you can improve the survey.
- **Pay Quickly**. It annoys Turks when you don't.
- **It's addictive**. It is hard to keep yourself from constantly checking to see if new results have come in.

Conclusion

Patrick's search for a product name demonstrates a creative approach for Mechanical Turk. He invested a few hours and around $20 and wound up with a name he knows is a good choice for his market and a nice list of prospects to notify about his launch.

The reason I like his approach comes back to my desire to measure and quantify marketing. If there's one thing wrong most people miss about marketing, it's measuring

and tweaking everything: your message, your copy, your videos, your tagline, even your price.

And while you can't easily tweak your product name, Patrick took the next best approach by measuring in advance instead of thinking about it for week and going with a gut feeling. Quantifying these kinds of things will improve sales and, trust me on this one, will allow you to sleep at night.

Original Location

http://www.softwarebyrob.com/2009/10/29/crowdsourcing-your-product-name/

How to Compete Against Open Source Competition

At some point in the past year I watched a video of an Eric Sink presentation and he asked the following question (he said it was asked of him by a college student):

Why would someone buy your product when it has an open source competitor?

He didn't answer it in his presentation, but if I remember correctly he said that to a college student, who tends to have a lot of time and little money, a product like Eric Sink's Vault (which runs $299/license) is an enigma.

Why would someone pay $299 per user when there's a perfectly suitable alternative *available for free?!*

Time vs. Money

At one point during college I had $6 in my checking account. This lasted for about three months.

I remember riding across campus, about a 15 minute ride (each way) to save $.50 on a slice of pizza. This is inconceivable in my life today. Back then, time was abundant and money was scarce.

Then I graduated and got a job. At a salaried job making $80k plus benefits your time is worth around $55/hour. Suddenly that ride across campus to save $.50 doesn't seem like the smart money decision it once was.

And thus it is with the majority of open source software:

Open source software is free if your time is worth nothing.

I'm bracing my inbox for emails from disgruntled Gimp users explaining how charging for software is bad, commercial software is evil, and my mother dresses me funny. But I don't buy it.

I've used mainstream image editors like Photoshop, Paint.NET and Gimp; some of my best friends are mainstream image editors. And when I saw Gimp I almost went blind. Children were weeping; fruit was bruising. The UI could kill small animals.

Are there exceptions in the open source world? Absolutely.

When an open source project gets enough talented people working on it, it can become a downright masterpiece.

Firefox rocks. WordPress is awesome. Paint.NET rules. And Linux is pretty cool, though the lack of drivers and ease of use as a mainstream desktop OS after all this time is still a disappointment.

And yes, I know about Ubuntu. I also know that every friend to whom I've recommend it has run into major compatibility issues or complete lack of drivers. Ubuntu is free, after 6 hours of research and command line tricks trying to get your laptop to connect to your network.

Have you ever tried Gimp? Or the admin control panel in Zen-Cart? Or tried to install a Perl or PHP module that didn't come out of the box? I've been a web developer for 10 years and I cringe when I see that I need a module that's not included…there goes two hours of my day searching, configuring and installing dependencies.

As a developer I've probably had contact with 300 open source projects, components, and applications. I estimate 80% of them required substantially more time to install, use, or maintain than commercial counterparts.

But depending on the price and feature set of their commercial counterparts, sometimes it's worth using the open source app and sometimes it's not. $299/user for Vault vs. $0/user for Subversion? It depends on how badly you need live support and guaranteed bug fixes.

The Differentiators

The areas that kill the most time when consuming open source software are:

- Installation process
- Documentation
- Support
- Usability

I'm sure we can all point out a handful of open source projects that have decent documentation and decent usability. The vast majority do not. Even fewer can be installed in five minutes or less, even by an experienced software developer.

How to Compete Against Open Source

As a commercial software vendor you have to focus on your key advantages over open source software:

- **Save Your Users Time.** Ensure a painless installation process, top notch documentation, top notch support, and a minimal learning curve for getting started using your application.
- **Market Hard.** You have a marketing budget; odds are high your open source competitor does not. If you can position your product well and build a reputation for good documentation, support and usability, you will sell software.

- **Focus on Features for Your Demographic.** Your open source competitor is going to win when it comes to college students, hobbyists, and other groups where time is worth a lot less than licensing cost. You will have an edge with business users since time is highly monetized for entrepreneurs and enterprises. Build features for people who are likely to buy your product.

You will find more success focusing on your strengths rather than your weaknesses.

Remember: while you're not going to win a feature race against an open source competitor, you'll do even worse in a price war.

Original Location

http://www.softwarebyrob.com/2009/08/11/how-to-compete-against-open-source-competition/

Expenses You Don't Think of When Starting a Business

I ruffled a few feathers with my recent post *The Software Product Myth*. The unrest surrounded my statement that making $2500/month from your software product wouldn't allow you to quit your day job.

The comments here and on a few social bookmarking sites mentioned that you *could* quit your day job if you wanted to, and that you *could* live on $2500/month just fine in many cities in the world (although in my hypothetical situation I was speaking about a developer based in the hypothetical U.S.).

We could get into a discussion about how much developers make, and how many costs you will take on by quitting your day job, but it's completely irrelevant.

The Point

The point of *The Software Product Myth* is that at some point you are going to have too few sales to support yourself monetarily, yet too much work to fit comfortably into your evenings and weekends. Whether your number is $1000/month, $1500/month, or $5000/month has zero bearing on that point…what matters is that building a product is a lot more difficult than most people make it out to be.

With that said, one of the helpful points that came out of the discussion is how many expenses you encounter when starting a company that you never knew existed.

Remember that line item on your paycheck that said something about retirement matching?

Or the disability insurance your company offers that you never knew they paid for?

Yeah, those are going to hurt.

You can go without these expenses for a short time while in startup mode, but if you plan to build a company that's sustainable in the long-run you're going to need to cover these expenses before you think about collecting a salary.

The Expenses

In putting together this list I looked through some of my old posts and also scanned my recent bank and credit card statements. I'm amazed at how many business costs I pay throughout the course of a year.

Depending on your country of residency these may not apply to you (people with national health care – consider yourself lucky!), but most of them apply in one form or another throughout the world. In addition, it is unlikely you will need every one of these expenses, but the intent is to be as close to an all-encompassing list as possible.

Core Business Expenses

- **Business Filing Fees** - This includes your business license, fictitious name statement, and reseller license fees (if applicable). They tend be paid annually and vary widely, but for a sole proprietorship (in the U.S.) you're looking at around $100/year. L.L.C.s and Corporations range from a few hundred dollars into the thousands.
- **Accountant** - Business taxes, especially if you have a home office, are easy to get wrong. As I said in *The Five Minute Guide to Becoming a Freelance Software Developer*, an accountant is not an optional expense. Costs range from $300-$1000 per year.
- **Lawyer** - Lawyers I've worked with run $250/hour and up. If you want contracts that hold up in court you're going to drop serious coin with our friends in the legal profession.
- **Liability/ E & O Insurance** – Varies widely, typically from $1500-$2500/year.
- **Health Insurance** – In the U.S., decent insurance for a family of three is now hovering around $800/month. It's less if you're single.
- **Disability Insurance** – It varies, but typically runs $250-500/month in the U.S. You may think this is optional, but consider that during any given year you are 4x more likely to become disabled than to die.
- **Life Insurance** – While you probably don't need it if you don't have a family, it's a good idea to have if you're married (and I would argue a requirement if you have children). If you're young and go with term life insurance you'll pay $10-20/month, but as you age that will increase dramatically into the hundreds.
- **The "Employer" portion of Social Security (FICA) and Medicare** – Often called the "self-employment tax," it eats up an additional 7.65% of your gross income if you're self-employed (since your employer usually picks up this portion).

- **Retirement** - Save for a rainy day. No one's matching your 401(k) anymore, and you should be putting away at least 10% of what you make.

Technical Expenses

A few of these apply only to companies that build software, but most apply across the board.

- **Windows Hosting** - You can go cheap, but you'll pay in other ways (don't say I didn't warn you). Decent hosting with decent support will run $20/month. I recommend *DiscountASP.NET* (even though they have a terrible website). They are responsive to support issues and very developer-centric. They roll out new .NET frameworks and while they are still in beta, and they are inexpensive considering the service and uptime.
- **Linux Hosting** – The same sentiment as Windows Hosting, but $10/month will serve you pretty well. As always, I recommend*DreamHost*, even with my recent blog issues. Did I mention the issue turned out to be a WordPress plug-in?
- **Bug Tracking** - You can go open source and save money, but you may lose it eventually in the time you spend maintaining and upgrading it. This one's your call. I've chosen to outsource my bug tracking to *FogBugz*, and it runs me $25/user per month. Pricey, but based on my hourly rate it's cheaper than the open source solution I used previously when you factor in upgrades, crashes, and manual workarounds for missing functionality.
- **Source Control** – This is one place where you can probably get away without spending any money, but I wanted to mention it anyway. As I discussed in *Source Control for Micro-ISVs*, I use*DreamHost* for hosting my Subversion repository, as it comes free with their base hosting account.

- **Advertising** - This will vary widely, but a decent Adwords budget will run $100-600/month (though it should be paying for itself).
- **Graphic Design** - Here's another danger zone where developers cost themselves money by trying to design their own graphics. Please, I beg you, pay someone to design your web site.
- **Phone** - $20/month for a land line. $50-100/month for a cell phone with a decent chunk of minutes.
- **Internet** - $20-60/month.
- **Fax Service or Fax Machine** – It seems like it should be brought into the back yard and shot, but I still send a couple faxes each month. *eFax* will run you $14/month (annual plan) or if you have a land line you can fork over $50 for a fax machine (or buy a multi-function printer).
- **Printer** - A color laser will run you $300-600 these days (I love my new *Samsung Clx-3175fn*). If you're fine going old-school you can get a B&W laser for around $80. Either way, the toner is what kills you. Set aside $50-150/year for toner and paper.
- **Computer** – If you're writing software you're probably upgrading your PC every 2-3 years. Figure $1,500 for a new laptop, $700 for a desktop, give or take a few hundred.
- **Software** - If you're one of those lucky Ruby or PHP developers then most of your tools are free. If you work with .NET be prepared to pay $1000-2000 for an MSDN subscription (or if you're developing a product talk to *Bob Walsh*, get signed up with Microsoft BizSpark and get the same thing for a few hundred bucks).

Based on the above, you can probably see how $2500/month isn't going to keep you in the lifestyle you're accustomed to…it'll barely keep the doors open.

Of course, this is far from an exhaustive list. If you have additional items that have caught you by surprise as you've started your business, please post them in the comments.

Update: If you disagree with one of my estimates, please don't post to the comments or email me. I've received a lot of email with information about using $1 Linux hosting, $1200 laptops, $100 printers, etc… I already know about these options, but saving $9 per month has absolutely zero impact on the point of this post, and as I said above, if you go with cheap hosting/laptops/printers "you will pay for it in other ways."

Original Location

http://www.softwarebyrob.com/2009/01/07/expenses-you-dont-think-of-when-starting-a-business/

The Entrepreneurial Mindset

Five Reasons You Haven't Launched

Photo by *stevendepolo*

This post is an accusation. A call to arms. A sharp stick that says "get off your ass and make something happen."

But I didn't write it for you; I wrote it for myself. Every one of these reasons has haunted me at one time or another over the past 10 years. Many a moon ago I thought I was the only person who struggled with them. Now I have several conversations a week that indicate otherwise.

These reasons will come to life every time you start something new, be it an application, a website, a book or a presentation. Excuses don't discriminate based on what you're creating.

So with that, here are five reasons you (and I) haven't launched…

Reason #1: You're Still Trying to Find the Right Idea

Give yourself a month. If you spend a month of "thinking" time, interspersed with a few hours a week researching ideas and you still haven't settled on one, close up shop.

Keep the day job. Hang around the water cooler. Become a lifer.

If you can't find a worthwhile niche in 30 days of intense thought and research there is trouble ahead, sir. This is an important decision, and yet it's just one on your path to launch. There are thousands more that need to be made before you'll get there.

If you can't make this decision in 30 days, save yourself the time and aggravation of trying to launch a startup.

The clock starts today.

Reason #2: You're Set on Doing Everything Yourself

Yep, I am going to say it again.

One of the most *time consuming startup roadblocks* is your need to control every detail and do every piece of work yourself. When you're scraping together 15 hours a week of night and weekend time, outsourcing 5 hours a week makes you 33% more productive.

You may be able to slice PSDs, but there *are people* who can do it faster and better than you. For $159 save yourself several hours of time.

You may be able to write web copy, but *there are people* who can do it faster and better than you.

Depending on your skillset, the same goes for graphic design, theme creation, creating unit tests, and a slew of other pre-launch tasks.

Time is one of your most precious commodities. Conserve it with a passion.

Reason #3: Hacker News, WoW, and [Insert Distraction Here]

Distractions are everywhere, including new-fangled distractions that disguise themselves as productive work (think Twitter).

Forget the TV and video games, how many times have you found yourself thinking you were being productive only to look back and realize you spent 3 hours searching and evaluating something you may not need until 6 months down the road? And frankly, you'll probably never need it.

Another common distraction masquerading as productivity is reading business books. I've already *beaten this one to death* so I'm not going to re-hash it here. Suffice to say, most business books are the entrepreneur's Kryptonite.

Avoiding distractions takes discipline, and discipline is hard. Especially when building your product is supposed to be fun.

Wait, this is supposed to be fun, right?

I hate to be the bearer of bad news, but maybe building a product isn't quite what most blogs, books and magazines make it out to be. Maybe it's actually a long succession of hard work, late nights, and a boatload of discipline.

For some people that's fun. For others, not so much. You should decide which camp you're in before you get started.

Reason #4: You're Busy Adding Features (That No One Will Use)

When was the last time you spoke with someone who is planning to use your application in the wild?

Not your friend who's testing it out to make sure it doesn't crash when you login, but a real customer who entered their production data and told you they are anticipating the release of your application more than Iron Man 3?

If you've not working directly with at least two customers you have no idea if what you're building is adding value. Or a total waste of time.

Reason #5: You're Scared

We all are.

In my experience, fear is the #1 reason that keeps people from launching. Except most don't recognize it as fear because it manifests itself in other ways. Needing to make everything perfect, to add one more feature, or to read one more marketing book are all ways that fear turns itself into excuses.

This is how fear really works. It keeps you from launching by tricking you into thinking you have real work to do, when that work is actually pointless busy work created to stave off your launch. Because launching is scary.

Actually, it's *terrifying*.

But we're all terrified at one time or another. You just have to deal with it and move on or you'll never get your product out the door.

Bonus Reason: You Have Launched, But No One Noticed

You had a great idea for a product. You went and built it in your bedroom. And now it's available for sale. Hooray! You made it to launch. Only problem: no one noticed.

You missed one minor detail in your mental business plan. You need a targeted, sustainable strategy for bringing prospects to your door. Or in this case, your website. If it's not both targeted *and* sustainable you are out of luck.

You can have a great launch day with a link from TechCrunch and an article on Mashable. But then it goes away. The traffic dies. 10,000 uniques the first month, 500 the second. 300 the third.

How can you build a business on 300 unique visitors a month?

The answer is: you probably can't. You need to figure out how targeted prospects are going to find you consistently, month after month. Because 10,000 unique visitors in a month is a nice bump in sales…nothing more.

What are your best bets? I know you want me to say "fun" and "sexy" things like Facebook and Twitter, but alas it's the un-sexy things like SEO, building an audience (blog or podcast), and having an engaged mailing list that work over the long-term.

(I have a lot to say about these un-sexy strategies. If you want more info check out my *startup book* for an entire chapter on this topic).

Your Turn

This list is incomplete. What have I missed? What are the reasons you haven't launched? Post them below and let's discuss.

Original Location

http://www.softwarebyrob.com/2010/11/10/five-reasons-you-havent-launched/

It's Easy to Be Great…It's Hard to Be Consistent

The title of this post comes from the book *Born Standing Up: A Comic's Life*, Steve Martin's memoir about his days of stand-up comedy. It's a crazy startup-like story that details his 14 year ascent to the top of comedy, followed by 4 years of wild success playing sold-out arenas; an unprecedented feat at that time.

Among the sage-like insight Steve imparts in the book is the observation that many performers have outstanding shows now and again. But very, very few are able to

consistently nail their performance. Steve proposes that it's consistency that makes someone successful in the long run, not having a great show now and then.

You're probably wondering what this has to do with your startup.

So let's get to that part.

Your Tenth Appearance

One of Steve's diatribes surrounds the number of times you have to be on The Tonight Show before people recognize you in the street. In his early career he believed that appearing on The Tonight Show would be his "big break" that would set him up for the rest of his career.

In other words, it was his "link from TechCrunch."

Steve says that after your first appearance no one recognizes you.

After your third time people start to think you might look like someone…but they can't figure out who.

After the sixth time they start to know you from somewhere…maybe you work at the post office?

And after the tenth time they start to recognize you as someone who might have been on TV. But they have no idea who the hell you are.

After 10 times on The Tonight Show.

There is no "big break." Not in comedy, and not in startups.

No link from TechCrunch, write-up on Mashable, mention by Leo Laporte, or Tweet by Ashton Kucher is going to make your business. Each of them will send some traffic your way and a very small portion will "stick" if you have the right mechanisms in place (you want email and RSS subscribers, not drive-by visitors).

But forget the fairy-tale notion that you're going to have a "big break" after which everything will be a walk in the park.

The Madness

Why is it so hard to be consistent? Because of something called *The Madness*. (A term coined by the *TechZing* guys during one of my *recent appearances*).

The Madness is the feeling you get when you have a new idea. The insatiable urge to start researching, marketing or coding. The unquenchable, zombie-like quest to turn your idea into something tangible.

During the time you're infected with *The Madness* you can be more productive in 12 hours than you were in your previous 40.

Talk about getting into the zone…you go days without showering because you just need to get one more part of your idea completed (oh wait…I'm the only one?). *The Madness* is amazing for productivity!

But the problem: *The Madness fades*.

Sometimes it comes and goes in a few days, other times you can keep it around in a more manageable form for months (meaning you take showers). But at some point every one of us loses interest in a project.

We're entrepreneurs – we're made to be passionate about ideas, and the*next* idea is always the easiest one to be passionate about.

Yep, it's tons of fun to think of ideas. And it's easy.

It's also fun to start building them. And still relatively easy.

But finishing the implementation of an idea? Seeing it through the 4-6 months (or more if you're playing with fire) until it turns into a real product? That's when it starts to get rough.

You will inevitably start to question if the idea is any good at all, and why you started working on it in the first place.

Then there's your launch, supporting it, improving it, and building a consistent, ongoing marketing effort that brings in a sustainable flow of people interested in buying your product. This is when it *becomes drudgery*.

This is the point where most who've managed to make it this far throw in the towel.

Wrap Up

It's easy to come up with great ideas, and it's a breeze to start coding them. It's only slightly harder to turn them into a functioning product.

But it's extremely hard to consistently focus on an idea over months or years. Very few people have the discipline to do it.

This is not a lesson reserved for startups. Look at all the bloggers who's come and gone over the past 5 years.

Think of every post that hits the front page (or even the #1 spot) on Hacker News (or Digg or Reddit). Many were probably better writers, had more insight, and better ideas than the big name bloggers you read. But publishing a post every week is hard work, and consistency trumps that one or two great posts someone wrote back in 2008.

It's easy to be great…it's hard to be consistent.

What Can You Do?

I can't end this post without some practical steps that can help you maintain consistency. I've mentioned a few ideas in previous posts and I'll state them here again:

Give Yourself a Cooling Off Period

(from *Lesser Known Traits of Successful Founders*)

I have a rule that I never spend money on an idea in the first 48 hours. During this time my judgment is clouded by the euphoria of having this amazing new idea. Given that

I've had several hundred ideas over the past few years, at a minimum I've saved myself a few thousand bucks in domain registration fees.

When you have an idea, write it down and come back to it in 2-3 days. You should have a better sense of its true merits after giving yourself time to cool off.

If you do your research and decide to pursue it you should stand a better chance of making a decision based on logic rather than emotion, which will ideally mean you'll have a better chance of sticking with it in the long-term.

Find Accountability

(from *Why Startup Founders Should Stop Reading Business Books*)

Find some kind of accountability group or *startup community*. If you can do it in person all the better, but I'm now starting to see accountability groups forming over Skype, as well.

Hit *Meetup.com*, and search Facebook for other tech entrepreneurs in your area. Find like-minded startup founders and get to know them.

Then hand pick 3-5 others and organize a small, private, committed accountability group that meets every 1-2 weeks where you can discuss your projects. You'll be amazed at the impact this has on your motivation, your consistency and your overall success.

Other Ideas?

If you have another approach you use to maintain consistency please share it in the comments.

Original Location

http://www.softwarebyrob.com/2010/11/04/its-easy-to-be-great-its-hard-to-be-consistent/

Lesser Known Traits of Successful Founders

I've worked with several hundred entrepreneurs over the past few years, and after the first 50 or so I began to notice a pattern. There are traits that successful founders possess that tend to be weaker or absent in people who are never quite able to make it work.

There are obvious traits that you need as a startup founder: a strong work ethic, perseverance, a desire to learn, etc… But at this point we're all bored to tears of these entrepreneurial generalities. If you need someone to tell you to work hard and persevere, you should cut your losses now and head back to your cubicle.

So today I want to focus on a handful of lesser known traits that will contribute to your success as a founder.

How Quickly Do You Respond to Opportunity?

Some people will contemplate a course of action for months before taking action. Others will do it in a matter of days.

Transitioning from an idea to implementation should not take months. If you can take an idea, perform the proper research, and make a swift but logical decision to move forward or cut bait, odds are better you're going to succeed in the long run. Maybe not with this idea, but at some point in the future.

The main reason for this is that wasting time on over-analysis is just that…a waste of time. Successful founders don't tend to waste time. They decide quickly and get things done. Productivity comes not just from effort, but from deciding to begin that effort months before the other guy.

This can be taken to an extreme and become a weakness, of course. Some people react too quickly; spending 5 minutes on research when they should spend 5 hours, and bouncing from one idea to the next. So with quick response must come the ability to see an idea through to its conclusion.

How Flexible Are You?

Are you dead-set on a specific product idea? Or hell-bent on customers using your product in a certain way? These are both strikes against you.

As you research and ultimately build your product, you're going to find you need to make major changes not only to the product itself, but to your vision of how people will use it. If you are unwilling to make these changes your venture is less likely to adapt to customer needs.

Flexibility is hard, especially since most of us tend to hold tightly to the initial vision of our product. This is almost by necessity; it's the only way some of us can keep from throwing in the towel. But that commitment to your idea will need to change as you move forward.

Think about it this way: if your market asked for it, would you be willing to switch from a web app to an iPhone app? Or from iPhone to desktop?

How about switching from building a product to providing services? Monthly to one-time billing? Or from selling your application to open-sourcing it?

If the business justified it, would you be willing to make a drastic change?

Do You Have Confidence in Your Ability to Execute?

Notice the question is not "Can you execute?", it's "Do you have confidence in your ability to execute?"

One thing you have to get used to as a founder is the ongoing stream of setbacks. Development *always* takes longer than you think, you *always*have more bugs than you'd like, your mailing list is *never* as large as you want, and on and on.

When you have high hopes for your product, everything seems like a disappointment.

Those who have a high level of confidence in their ability to execute and are able to look past these setbacks, are more likely to succeed. Confidence is not easy to come by,

but those that have a realistic self-image (not out of whack in either direction), stand a far better chance of creating a successful company.

Since "confidence" is a hard concept to evaluate or even understand, I'm reading an excellent book on this subject called the *Six Pillars of Self-esteem*. I realize it sounds like a cheesy self-help book, but it's not. It's written by a researcher who has spent his entire career studying what motivates people and contributes to their success in life.

Highly recommended, even if you already feel like you have this one knocked.

Can You Focus Long Enough to Deliver?

We all know the guy who moves from one idea to the next and never finishes anything. He's freakishly smart, but leaves a trail of half-finished carnage in his wake. Staying focused is a huge part of being successful.

To evaluate your level of focus, look at your history. How many half-finished apps are sitting on your hard drive? How many blogs have you started and abandoned within three weeks? How many have you worked on until they were done?

People who have trouble staying focused on an idea often feel intense passion for it early on, obsess about it for a week or two, and burn themselves out on it by the one-month mark.

If you have trouble with this you may need to give yourself a cooling off period. What often happens is you become so engrossed in the idea that you never stop to look at it rationally and realize a glaring flaw. A flaw that you find 2-3 weeks later when you realize you probably won't be able to pull it off after all. Things that would have been nice to know 2-3 weeks earlier.

I've found that the first several hours (or even days) after coming up with a new idea are filled with irrational, euphoric thoughts of how easily it can be executed and how well the market will receive it. You'll often hear yourself saying "Why hasn't anyone thought of this?"

I have a rule that I never spend money on an idea in the first 48 hours. During this time my judgment is clouded by the euphoria of having this amazing new idea. Given that I've had several hundred ideas over the past few years, at a minimum I've saved myself a few thousand bucks in domain registration fees.

When you have an idea, write it down and come back to it in 2-3 days. You should have a better sense of its true merits after giving yourself time to cool off.

If you do your research and decide to pursue it you should stand a better chance of making a decision based on logic rather than emotion, which will ideally mean you'll have a better chance of sticking with it in the long-term.

Original Location

http://www.softwarebyrob.com/2010/09/16/lesser-known-traits-of-successful-founders/

The Terror of Firsts

The first time you try something it's scary.

Some "firsts" I've experienced in the past 10 years that have put some serious fear in me:

- my first face to face with a potential client
- my first published article
- my first blog post
- my first product launch
- my first product acquisition
- my first time speaking at a user group
- my first time speaking at a conference

As humans we fear the unknown. We fear failure…rejection…mistakes. These are either feelings that come naturally or have been pressed into us by society.

This fear is what makes starting a company hard. It entails large amounts of risk and there's so much potential for failure, rejection and mistakes. In fact, most of these are almost inevitable if you choose to start something that makes a difference.

It could be a blog, a consulting firm, a product, a company, or simply an attempt at speaking in front of a group. But it's all terrifying that first time.

What's odd is how much easier it is the second time. And the third. And how, by the fourth time you can't even feel the hair on the back of your neck, or the sweat in your palms. The terror goes away surprisingly quickly. Then you have to find the next experience that creates that fear in you.

That up-front fear is a big indicator that you're going to grow as a person if you proceed through it.

Five years ago I re-read and re-published my first blog post at least 20 times. No one was reading, but it felt insanely scary. Now it's akin to brushing my teeth.

The terror wears off pretty quickly

The interesting thing I've found is that the more taste of this growth you have, the more you want it. It's a rush, and it's addictive.

It's a huge confidence boost to look back at the things that scared you last month, last year, or even five years ago. And realize you conquered them.

This kind of success leads you to trust your instincts. It builds confidence. It eliminates pointless analysis that used to keep you spinning on decisions for hours, days or weeks.

Overcoming the terror of firsts is hard, but it's what makes the goal beyond the terror worth achieving. The terror that stands between you and the goal is something 99.9% of people will never overcome.

Are you part of the 0.1%?

Original Location

http://www.softwarebyrob.com/2010/04/14/the-terror-of-firsts/

Why Startup Founders Should Stop Reading Business Books

You're a startup founder, whether actual or aspiring. You're a constant learner; a student of life. Also a student of reading.

You own a wall of books, or perhaps a Kindle or iPad bulging at the edge of its hard disk with non-fiction you've earmarked for future reading. Business books are a founder's learning tool. A way to stand on the shoulders of giants.

So let me tell you why you should stop reading them.

The Gladwell Effect

Every entrepreneur has read at least one book in the "*Gladwell/Godin*" genre. These books are fun to read, focus on a single high-level premise and are packed with entertaining anecdotes to demonstrate or support that premise.

(Confession: I've been Seth Godin fan-boy for years. I am completely stoked that we are speaking at the same conference in October.)

The problem is not that these books are a series of *anecdotes disguised as science*. The problem is that the premise of all of them – whether true or false – is irrelevant to you as a startup founder.

The knowledge that you need 10,000 hours to master a subject, that certain trends become viral after the 412th person adopts it, or that you should make your product remarkable, is not going to help you launch. Nor will it save you a single minute during product development, or sell one more copy of your application.

"But wait a minute," you say "isn't knowing about tipping points and purple cows going to help me in the long-run as I run a business. I mean come on, I need to know about marketing and trends and stuff, right?"

There's a slight chance it will. But probably not. There are actually two problems with this line of thinking:

- The amount of actual information garnered from this kind of book can be summarized in a page or two of written text.
- The information is at such a high level that it's downright impossible to implement. Knowing you need to make your product remarkable is one thing. Knowing *how* to do that is another thing entirely.

How many times have you finished one of these books and thought: "Great…but what do I do now?"

The "Everyone is in the Fortune 500" Effect

Ok, so high-level, feel good books about social/marketing trends are not going to help my startup. But what about other kinds of business books? Books that deal with real data and have actual case studies?

I was on *paternity leave a few weeks ago* and I had time to catch up on reading a long list of books that have sat stacked on my desk for months. These books are more specific and rigorous than your standard Malcolm Gladwell fare; things like *Getting to Plan B*, *The New Business Road Test*, and *Business Model Generation*. I've read similar books like this in the past as well, some of my favorites are *Good to Great* and *Built to Last*.

As a serial entrepreneur these books should be right up my alley. They cover topics like how to pivot your product and/or revenue model to find market fit, how to test a business idea before building the business, how to find a business model for your company, and how to build great companies.

And these books are great at doing just that…if you're Sony. Or Proctor and Gamble. Or Panasonic.

The thought that kept running through my mind as I read them was:

> *"This information would be helpful if I needed to generate a huge business plan to impress a business-school professor."*

The problem here is also twofold:

- Many of these books are weighed down with so much theory it's like sitting in a horrific MBA course from the 1950s (the first three I mentioned are in this trap, the last two not so much)
- They speak to markets and business opportunities measured with 8 zeros or more. Massive markets that you don't need to understand to run a software company.

Before you know what kind of company you want to start this can be helpful. Think of it as a survey course you take in college that shows you all of your options for starting a company.

But if you're past the point of needing a survey course, and you've settled on a focused type of entrepreneurship involving self-funded, organically grown companies, you will find the vast majority of business books have no relevance to this path.

Even if you're going after angel and VC funding, your choice of books that actually apply to your market instead of the one billion people who will buy toothpaste next year, is slim.

Once you've decided to start a startup and you know if you're going to self-fund or raise capital, you need laser-focused books (or better yet blogs, podcasts and mentors) that speak in-depth about your particular situation.

Starting a self-funded startup but listening to how a bunch of venture-backed companies made it is a waste of your time. Inspiring stories aren't going to help you unless you're

in desperate need of inspiration; you need knowledge that comes from someone with actual experience.

The Information Overload Effect

Finally, if you find a book that has some tidbits of semi-actionable information that you're not ready to implement in the next month, think really hard about investing the time to read it.

Given the amount of information we consume every day, the portion of the info that you can retain, synthesize, and put into action is plummeting.

The problem is that we're addicted to consuming. Addicted to the fire hose of tweets, blog posts and books because it makes us *feel* productive and informed. When what they really do is kill time.

Reading a business book that does not have a direct impact on what you're going to be working on in the next 1-2 months is about as productive as *watching Lost*. Indeed, I would go so far as to say that:

For entrepreneurs, reading business books is the new television.

Objections

Here are two potential objections that I'd like to address:

Objection #1: "I like reading Seth Godin because everyone's talking about it and it makes me feel informed."

Great. Me too.

But instead of spending hours reading the book, read a detailed review of the book, or listen to an interview about the book. Trust me, you will pick up all of the salient points in 30 minutes. The only thing you'll be missing out on are the copious examples.

I've done this with the last 3 Seth Godin books, and the previous 2 Malcolm Gladwell titles. It works amazingly well.

Objection #2: "I like reading business books as a hobby."

Then by all means do this in your spare time. But know that it is a hobby akin to stamp collecting or gardening; it makes no contribution to your productivity as a startup founder. If you think that reading a Seth Godin book will ever put a single item on your task list that will help your startup, you are mistaken.

"Ok, I buy this preposterous idea you're proposing…what should I do now?"

Let me re-iterate that I'm not saying you should stop reading books. What I'm saying is that once you've determined your area of focus, you should spend the vast majority of your time reading books in that genre, instead of general business/entrepreneur fare we all seem to get sucked into.

In other words, look for resources that provide actionable take-aways to improve your business and that focus on topics that are specifically relevant to your situation. You want laser-focused, *just-in-time* learning.

This is the reason I focused my *startup book* on a tight niche instead of giving it broad appeal, even though it means I will sell fewer copies.

I narrowed the focus because it allowed me to speak directly to my audience, and to be ultra-specific about tools and approaches instead of copping out and using a stack of generalities that provide little value to the reader. And I think I made the right call.

I've received numerous emails with the same basic sentiment: "I've taken away more action items from this book than any other startup book I've read." This was my goal, and one I could only accomplish by focusing on the small niche of developers looking to launch startups.

With that said, here are a few examples of the types of books I recommend reading. Books that offer specific approaches that you can put into practice immediately (assuming you have launched or are launching soon). Any of the following will provide a 100x return on your time investment compared to the titles I've mentioned above:

- *Web Copy that Sells*
- *Always Be Testing*
- *Web Analytics An Hour a Day*
- *Web Design for ROI*

Moving Beyond Books

One final thought. As I've traveled this entrepreneurial journey I've noticed that my two biggest sources of learning have been my own first-hand experience, and the knowledge of other entrepreneurs.

So my first piece of advice is to *get to work on your startup*. Your code isn't going to write itself, and every hour of first-hand experience you gain launching your own product means you are better equipped to deal with the craziness that is a startup. No book can teach you that.

My second piece of advice: Find some kind of accountability group or *startup community*. If you can do it in person, all the better.

Hit *Meetup.com* for starters, and search Facebook for other tech entrepreneurs in your area. Find like-minded startup founders and get to know them.

Then hand pick 3-5 others and organize a small, private, committed accountability group that meets every 1-2 weeks where you can discuss your projects. You'll be amazed at the impact this has on your motivation, your progress and your overall success.

Original Location

http://www.softwarebyrob.com/2010/08/05/why-startup-founders-should-stop-reading-business-books/

Warning: Software Startups are Not as Easy as Everyone Says

A few months ago I had a conversation with a successful friend of mine (by successful I mean 29 years old, self-employed, a six-figure income, and two houses in Northern California). He runs a software company.

We were discussing what it takes to be a software entrepreneur and in a moment of self-reflection he said "I'm not as successful as I anticipated. I thought success would be easier."

One of the Smart Kids

When you're young people throw compliments at you left and right.

Wash your hands, get a round of applause.

Tie your shoe, get a standing ovation.

Read the Guinness Book of World's Records cover to cover…well, that will get you beat up by a 6-foot third grader named Chuck…but you get my point.

When you're young, people tell you that success is easy to keep you from getting discouraged. They say anything is possible if you work hard enough.

As you get older, if you read anything outside of school you know more than your peers. You wind up in classes with prefixes like Honors and A.P. And you begin to feel smart.

At some point you see infomercials that talk about how easy it is to make money investing in foreclosures, your own business, and the internet. But their angle is obvious. (I'll give you a hint, it starts with "But wait, there's more!").

And you hear stories from your unsuccessful Uncle Ned that end with "…and he invented the digital squeegee and became an overnight millionaire. What I wouldn't

give to be him!" Of course, you don't hear about the thousands of hours the inventor spent working alone in his basement preparing for "overnight" success. That would ruin his story.

The Startup Conspiracy

Your parents, teachers, television, and Uncle Ned have driven into your brain that being smart and having an idea is enough to guarantee your spot on the cover of ~~Red Herring Business 2.0~~ *Fast Company*. There seems to be some kind of conspiracy that startup success is easy; that it's handed out like a lei as you step off the plane at the San Francisco Airport.

But it's not easy. It's really hard. It's a combination of enormous amounts of hard work, intelligence, and luck. Anyone who tells you otherwise is trying to sell you something. Even if it's just a magazine.

Your 2% Chance

If I read another story about Facebook Apps I'm going to punch someone in their Facebook.

It seems like every time I open my RSS reader I see quotes like this one (from a recent *Fast Company* article):

"Created in a week by two brothers from India, [their Facebook app] caught on like wildfire, with half a million users signing up in the first 10 weeks."

This is one of about 20 articles I've read on the Facebook Apps phenomenon. Each has numerous examples of apps built in a matter of days that now have huge user bases.

Super poke your friend!

Throw a sheep at your friend!

Throw virtual poo at your friend!

That's not luck, it's pure genius…really.

An enterprising developer reads these articles and thinks "I want to write an app that gets half a million users in 10 weeks. I'm going to do what they did!" So she spends a week in her basement and cranks out a Facebook application. When she uploads it she finds a few thousand users and not enough monthly ad revenue to buy a Frappuccino. Why is that?

Because the vast majority of the traffic (87%) goes to *84 of the 5,000*apps currently on the platform. That's less than 2%. Not one of the articles I've read has mentioned this ratio, or the enormous luck factor involved in creating a "successful" Facebook application.

Luck out the Wazoo

I can hear someone out there thinking:

"What about HotOrNot.com and PlentyOfFish.com? Their founders make money hand over fist, they don't appear to be exceptionally gifted, and didn't put in much hard work. It was easy for them…"

Hot or Not and *Plenty of Fish* are the poster children for generating huge sums of money with little work (if you're not familiar with them, they are two dating sites that became smash hits "overnight" and make their owners millions of dollars per year for around 10 hours of work per week).

If you listen to *their stories* and compare the amount of luck to the amount of skill and hard work it took for their sites to succeed…the amount of luck is astonishing. Not only was their timing absurdly fortuitous and their ideas not exceptionally creative, but the sites were quite easy to build. They both seemed to walk into some kind of weird anomaly in the fabric of time that happened…as best I can tell…exactly twice.

If you think all it takes to build a software company is a nice website, a pretty application, and a link from GigaOm…please reconsider.

The web is littered with applications launched by people who thought it would be that simple. They read about *Hot or Not*, built and launched an app, received a link or

two from an A-list blogger and when money didn't start rolling in they called it quits. Now they're selling their labor of love on Sitepoint.

Don't let this happen to you.

The Bottom Line

There are exceptions to the rule that *software startups are hard*. But if you plan to succeed with an idea like throwing virtual sheep, you're better off spending your 40 hours of development time as a paid consultant and buying lottery tickets with your earnings.

If you want a chance at succeeding you should stay far away from trying to duplicate the "one in a million" stories. You're chances of success are at least an order of magnitude greater with a boring idea like *bug tracking*or *invoicing* software than with the next "hot" consumer website. Just ask the zillions of free dating sites that launched after Plenty of Fish hit it big.

You can spend your time dreaming of success and thinking that if you just had one brilliant idea you'd make millions, grace the cover of *Sexy Startup Magazine*, own a vegetarian-friendly Ferrari, and have that beautiful trophy husband/wife. Or you can put your head down, become an expert in your field, work your ass off, and someday, *maybe*…you'll have a taste of success.

Original Location

http://www.softwarebyrob.com/2007/11/06/why-starting-a-software-company-is-not-as-easy-as-everyone-tells-you-and-why-facebook-apps-are-crap/

One of the Most Time Consuming Startup Roadblocks

This post is an excerpt from the Micropreneur Academy, an online learning environment designed to get one-person web startups from zero to launch in four months.

You Must Unlearn What You Have Learned

I'm in the process of buying a house. The listing for the house we made an offer on last week has 45 digital photos that I would like to share with family and friends, but they are hidden behind a questionable JavaScript interface.

Using my advanced knowledge of web hackery (i.e. View Source), I grabbed the list of each image URL and put them in a text file. And the following ten seconds made a huge difference in how I spent the next 20 minutes of my day.

I copied the first URL into my clipboard and began to paste it into my address bar when I (for the hundredth time) realized that this is exactly the kind of task that appears to produce something, but is completely rote and repetitive. It would be simple (and fun) to write a Perl script to do the fetching, but that would take around the same amount of time.

So I sent the task to my virtual assistant (VA). It took me exactly 90 seconds to get the request to him, and within 24 hours I had a zip file of the images. It took 20 billable minutes and at $6/hour the 24-hour wait was well worth it.

This morning I realized I needed an image for one of my websites, and a change to the CSS. I'm not a great designer but I could have designed something in about two hours. I could have also made the CSS change and tested it in a few browsers in about an hour.

Instead, I opted to send these simple tasks to someone else at the cost of $15/hour. I wrote up an email and the task will be done in the next day or two.

- Time spent: 10 minutes
- Time saved: 2 hours, 50 minutes

I just received from a woman named Amy who lives in Canada. She has experience with multimedia and podcast editing. I contacted her because the editing process for the audio versions of the Academy lessons takes 60-90 minutes per lesson. Two lessons per week is 2-3 hours that I'm going to outsource to her for $15/hour.

She will likely produce a better finished product than I can.

How Much Can You Really Gain?

These are trivial examples of what I call *drip outsourcing*, outsourcing small tasks as I perform my daily work. Drip outsourcing has become invaluable to my productivity.

If you total up the three instances above it only amounts to 6-7 hours. But I do this constantly, every day. Before I start any task I think to myself "Could one of my contractors possibly do this?"

Over the course of a month you can easily save 20-40 hours without much effort. These days I save 60-100 hours a month.

This ties back into a topic I've spoken about previously: when it comes to your product should you *build it, buy it or hire it out?*. While you don't have to (and should not) hire out every aspect of your product, I cannot imagine handling every detail yourself (or within your team).

The roadblock that so many entrepreneurs encounter as they try to launch is *thinking they, or one of their co-founders, has to perform every task necessary to get their product out the door*.

Just for kicks I'm going to spit out an incomplete list of tasks needed to take a web-based product from idea to your first week after launch. Here we go:

- Niche Brainstorming & Mental Evaluation
- Niche Evaluation
- Niche Selection
- Product Selection
- Product Architecture
- Functional Design
- Database Design
- Graphic Design*
- HTML/CSS*

- UI Development (AJAX/JS)*
- Business Tier Development*
- Database Development*
- Creating Unit Tests*
- Creating UI Tests*
- Manual Testing*
- Fixing Post-Launch Bugs*
- User Documentation
- Installation Documentation
- Sales Website Site Map Creation
- Sales Website Copywriting*
- Sales Website Graphic Design*
- Sales Website HTML/CSS*
- Sales Website Programming*
- Sales Website Payment Integration*
- Product Delivery (via email, link on site, etc…)*
- Setting Up Email List
- Setting Up Domain Name & Web Hosting
- Setting Up Email Accounts & 800 Number
- Setting Up Analytics
- Pre-Launch Search Engine Optimization
- Pre-Launch Pay-Per-Click Set-up
- Initial Social Media /Viral Marketing*
- Pre-Launch Video Marketing
- Pre-Launch Partnerships
- Launch Press Release*

- Pre-Launch Email Marketing
- Pre-Launch Blogging or Podcasting

And probably a few others…That's a list of 37 tasks ranging in duration from 2 hours to a few hundred.

You'll notice many have asterisks next to them. These are the tasks that will be easiest to outsource – the tasks that require a technical or common skill that's not specific to your product.

Outsource your product architecture? I would consider it only for small applications.

Outsource your graphic design and HTML/CSS? Every time…

Avoiding This Roadblock

The best way is to to start small, gain comfort with a contractor, and gradually increase the amount you outsource.

Outsourcing is a learned skill, and you're likely to screw it up your first time around. Start with non-critical tasks and be very specific in how they should be executed. At first it will seem like you could do the tasks faster than the time it takes to assign them, but as you get to know the person you're outsourcing to it will quickly begin to save you time. If it doesn't, then you need to look for a new resource.

Hiring a Virtual Assistant (VA) is a great way to get started with almost no financial commitment and a low hourly rate (around $6/hour overseas, $10-20/hour in the U.S.). I have a few VA's I use for various tasks, and I've had great luck finding them on *elance*.

Graphic design and HTML/CSS are also great ways to dive in. Graphic design is nice because it's not complicated and what you see is what you get. It's either good or it's not. I've found design to be much easier to outsource than programming.

Finding decent designers is a little more challenging – elance is a good route, but asking around is an even better approach. Unfortunately I don't want the designers I use to

become overbooked so I can't mention them here (I do provide contact info for all of the companies I outsource to inside the *Academy*).

So take a risk this month: outsource your first task and see where it takes you. When was the last time a single tool or work habit offered the opportunity to save 20-60 hours in a month?

Original Location

http://www.softwarebyrob.com/2009/07/14/one-of-the-most-common-startup-roadblocks/

The Single Most Important Career Question You Can Ask Yourself

By the time I was 13 I had been selling candy and comic books to my classmates for almost 3 years. Though I did quite well, I was itching to try something bigger, and that meant extending my reach beyond the walls of Math class.

This was the late 80s, so resources were limited for a 13 year old living in the country. I ordered all of the free information available in the *work at home* section of the Penny Saver (a free newspaper consisting entirely of ads), and started going to the library twice a week to read up on entrepreneurship. I was searching for a business idea that I could pull off at 13, and after literally hundreds of books, booklets, and information packets I decided to publish my own booklet on comic book collecting.

"Smart"

Since I was seven years old I've been an avid reader. I consumed 2 or 3 books a week during my childhood, including a large collection of "crazy facts" books and the Guinness Book of World's Records (every year). By the time I was 13 I'd been reading 2-3 books a week for 6 years, and the breadth of my knowledge was astonishing for someone my age.

I knew how the stock market worked, why Beta had lost to VHS, why Apple was losing market share to the PC, and how double-entry accounting worked (although I couldn't *do* double-entry accounting). But I had no idea how to start a business. With all of my book knowledge about the business world, I had no clue how to execute an idea.

I'd read several books on self-publishing and writing non-fiction, and I could have a *really* good conversation about them, but I'd synthesized the information for knowledge-sake, rather than to act on it. That made a big difference.

And so it continued for months…I didn't have the guts to start writing the booklet for fear I wouldn't know what to do next. Instead, I visited more libraries and sought magazines that did nothing but rehash information I already knew. I filled my head with the same information from piles of resources, but I still couldn't get things going.

After months I finally took the leap. I spent 60 hours researching and writing the booklet. I printed it at Kinko's, placed a few classified ads, and sold 9 copies at $15 apiece, breaking even on the cost of printing and advertising. Financially it was a wash, but I learned a valuable lesson.

The Question

13 years later I went through a nearly identical scenario when I started my consulting firm, *The Numa Group*.

I'd read stacks of book about entrepreneurship, startups, management, leadership, consulting, and running a service business, but I had no idea how to get started. I was waiting for the one book that was going to kick me into action by telling me exactly how to proceed given my skills, strengths, goals, and financial situation. Alas, that book never appeared. I finally realized that I needed to take a leap of faith and go to a place where no one else could lead me.

Through these experiences I realized the following:

> *Some people are consumers by nature; they consume vast quantities of knowledge purely for learning's sake. Others are producers; they consume*

> *knowledge with the intent of one day acting on the knowledge and producing something, be it a book, a song, a blog, a startup, etc… Neither is better than the other.*
>
> *The key is to answer one question:* ***which are you?***

I want to say again that neither is better than the other. If we were all consumers we wouldn't have anything to watch; if we were all producers no one would be reading our blogs or listening to our podcasts. What matters is:

There is a huge benefit to finding out if you tend to be a producer or a consumer.

Which Are You?

If you convince yourself that the zillions of books, blogs and podcasts you've consumed over the past 2/3/4 years are in preparation for that glorious day when you'll tell your boss to stick it in his ear because you're heading out the door to starting your own company, you are wasting your time. Don't read another blog post about startups, Micro-ISVs, or the business of software. Once you're done reading this post (or even before), go start building something (a product, a blog, a company…). Because until you do, the knowledge you're gaining is all but worthless to you.

But if you realize that one of the pleasures in your life is to read about code/startups/entrepreneurs/music, then embrace that you are a consumer. Knowledge for knowledge's sake is not bad as long as you realize that you are not working towards an end beyond your own edification, which again, is not a bad thing.

Likewise, if you're someone who has an unquenchable desire to *produce something*, then stop reading about other people, and start doing it yourself. Seriously, don't read another blog post, tweet, or issue of Fast Company until you've made a visible move towards that goal you so desperately want, but think that reading and dreaming about will somehow make it come true. Once you've made that single action towards advancing your idea, you can come back and read a few more posts.

While it's true it's not either-or, that you are likely a mixture of both types and will experience fluctuation in your ratio of production vs. consumption from one month to the next, unless your name is Robert Scoble you really do have to choose one or the other.

To recap:

- Consuming for the pure love of learning is absolutely ok.
- Producing purely because you have a fire that won't die until you do is fine, too.
- But don't kid yourself about who you are.

If you've been reading startup blogs for years and never started anything, it's time to accept that your tendency is to be a consumer. It's not to say you can't break out of that classification by starting something, but if you haven't done it thus far you're not likely to do it soon without some external motivation (maybe this post?).

If you have 50 software product ideas and your hard drive is littered with folders containing 30 lines of code from each, you tend towards being a consumer (or at least a producer who has trouble finishing things).

And if you figure out that you are a producer, stop daydreaming about the day you'll make things happen. Start making it happen in the next 30 days, or forever hold your peace.

Original Location

http://www.softwarebyrob.com/2008/05/18/the-single-most-important-career-question-you-can-ask-yourself/

Don't Plan to Get Rich from Your Startup

Getting rich shouldn't be your goal when launching a startup.

Thousands of people who are better developers than you, are better at evaluating markets, and better at marketing software have built products and still work for a living.

The Entrepreneurial Mindset

There is a chance that your startup will provide riches beyond your wildest dreams. A *slightly* better chance than buying a lottery ticket.

And while it's true that you might get lucky your first time, *"luck" is not a plan*.

During his presentation at the *Business of Software* conference in 2009, *Dharmesh Shah* said "For your first startup, maximize your odds of a modest outcome."

Have you ever considered aiming for a modest outcome?

This approach gives you time to learn the ropes. There are hundreds of things to learn your first time at the plate. That learning curve coupled with a big market and big competition is very likely to crush your chance of success.

Your first time at the plate you already have one things stacked against you: lack of experience. Don't stack the complexity of a big market/big competition as well.

Maximize your odds of a modest outcome by choosing a niche market. It's not as sexy as conquering the world with the next brilliant consumer-focused social media network microblog app, but your chance of a modest outcome is 10x…perhaps 100x greater.

Start your startup. Learn the ropes.

Then, whether you succeed or fail, you can swing for the fences. That second time you may just make it.

(Since several have asked…this is one of the reasons my upcoming book about starting a startup is titled *Start Small, Stay Small*)

Original Location

http://www.softwarebyrob.com/2010/06/10/dont-plan-to-get-rich-from-your-startup/

Paying the Price of Success

Micropreneurship (one-person entrepreneurship) is a fantastic experience. The first time someone pays you for software you wrote, your head will nearly spin off its axis.

And ultimately, if you're able to harness the power of leveraging software instead of time, you will achieve more freedom than you're probably ready for. The first time I took a month off with two days of notice I had the nagging feeling that I needed to do 173 hours of work when I got home…then I realized that hours and dollars no longer correlated.

It's hard to re-train your mind out of the dollars for hours mentality. For me it took well over a year.

The price to achieve this kind of payoff involves a huge up-front investment of time.

And it may involve a substantial financial investment acquiring products.

Or maybe you'll need to step away from the code, give up that control we all feel we need, and hire out some development so you focus on other areas of the business.

The price you pay is negotiable and is truly guided by your personal goals. But the bottom line is: you will have to pay a price one way or another.

It's a long road to becoming a Micropreneur. There will be many long nights, especially in the beginning. It's critical to know what you want out of Micropreneurship so you can make the right decisions along the way, and to give you something to hold onto when you're burning the midnight oil for the fifteenth night in a row.

Original Location

http://www.softwarebyrob.com/2010/03/18/paying-the-price-of-success/

How to Avoid the Three Startup Danger Points

I've communicated with hundreds of startup founders over the past three years, and I've begun to notice a pattern.

There are three points during the creating of a startup where the founders are most likely to close up shop. I call these the "danger points" and this post looks at how to avoid them.

Point #1: Choosing a Product Idea

There are three types of people: those who understand binary and those who don't. No, wait…that's a different blog post.

There are two types of people: those who are impulsive and those who over-analyze decisions. If you're impulsive don't worry about this point; you'll have no clue what it's about.

But if you tend to over-think your decisions, then choosing a product idea is going to take months…nay, years. That's right – odds are high that by the time you figure out what you want to build you could have built and launched multiple products in the same time frame.

This is because committing to a product feels like a permanent decision, and a decision that's easy to screw up.

While there is a limit to what you can "know" about a product idea before you start building, if you do your niche research you can eliminate much of the uncertainty that goes into choosing a product idea (Pamela Slim talks about finding a niche *here*, *here* and *here*, and Mike and I have a step by step approach for finding and testing a niche in the *Micropreneur Academy*).

While you'll never remove all uncertainty, you can remove enough that making the niche decision is a bit more clear cut than the dart board approach you're using now. Niche research can make a world of difference in your confidence level as you start writing code.

Point #2: Two Months Into Development

Around the two month mark you will hit a dip. A big one.

You've spent every evening and weekend for the past two months and you're barely making progress. And with the feature list ever-expanding it looks like you won't launch for 8 months (or more). This is a common cause of death for small startups.

The first way to combat this milestone is to have a detailed feature list and an estimate for every task on that list. This list should include marketing tasks and anything else you need to get through your launch date. This list will be large; likely 80-120 lines long.

With an estimate for each item you should be looking at 400-600 hours*total*. For everything.

If you're over 600 hours you need to cut something. Unless you have multiple, commited founders each working 10-15 hours per week on your product, you will not make it to launch if your total is over 600 hours (well, there's a chance you will make it to launch, but a very small one).

This means cutting features. It's a hard decision to make, but the quickest way to reduce your total hour count is not to "get faster at writing code" (I say with the tone once used on me by a manager), but to push features to v2.0.

The other approach I've mentioned before is to outsource. Even outsourcing construction of your sales website, or the HTML/CSS conversion, or copywriting can remove 20-80 hours from your time line and dramatically increase your chance of making it to launch day.

Point #3: One Month After Launch

The main reason someone shuts their company down after launch? They thought it would be easier to make sales.

They thought people would flock to their idea, fumbling for their credit cards the moment they heard someone had developed a re-tweeting cloud app for Facebook

fan pages. But alas, being heard above the din of the internet is a tricky thing. Ask any startup founder.

No, the first few months after launch are extremely hard unless you've planned your launch well and identified the number one goal of your website. It's not to sell your product…it's to get people to come back to your website at a time in the future when they are ready to buy. This is most commonly achieved through a blog, podcast, or email list.

This is off the radar for most developers – most of us just want to write code and sell it to people. Oy, if only it were that easy. (I've written an entire section on the "#1 goal of your website" in my upcoming book, *Start Small, Stay Small: A Developer's Guide to Launching a Startup*).

The bottom line here is to take your launch seriously, as seriously as you've taken your product development.

A successful launch will provide motivation to continue during the rough months ahead as you begin to support your product. Having an email list to notify on your launch day will result in your best sales day *ever*. This is the proper way to launch.

The wrong way is to assume that emailing bloggers the week before you move your website live will drive traffic that result in sales. Or that you'll make hundreds of sales using AdWords. AdWords are good, but you're not going to make a big splash on day 1. They take time to hone and become profitable. Same with SEO.

The marketing arsenal of a small startup should include all of the above items, but most of them take months (4-6 or more) to become profitable. If you want to turn a profit quickly…concentrate on building an email list for your launch.

The final aspect of launching well is having the right expectations.

A Micropreneur emailed a few months ago asking how he could sell six figures worth of his product in the first six months. He's been following the edge cases too long…

reading *Fast Company* and watching *Balsalmiq* (a great product, just a very atypical first year for a small company).

I let him know he would be better off if he adjusted his expectations and shot for $6k in revenue in the first 6 months. Sound low? It's more than the vast majority of launches I've seen in the past three years. But once you make it past that first six months, that's when things get interesting.

Original Location

http://www.softwarebyrob.com/2010/05/21/how-to-avoid-three-startup-danger-points/

The Software Product Myth

Most developers start as salaried employees, slogging through code and loving it because they never imagined a job could be challenging, educational, and downright fun. Where else can you learn new things every day, play around with computers, and get paid for it? Aside from working at Best Buy.

A certain percentage of developers become unhappy with salaried development over time (typically it's shortly after they're asked to manage people, or maintain legacy code), and they dream of breaking out of the cube walls and running their own show. Some choose consulting, but many more inevitably decide to build a software product.

"After all," they think "you code it up and sell it a thousand times – it's like printing your own money! I build apps all the time, how hard could it be to launch a product?"

Against All Odds

And most often the developer who chooses to become a consultant (whether as a freelancer or working for a company), does okay. She doesn't have a ton of risk and she gets paid for the hours she works, so as long as she has consulting gigs she can live high on the hog.

But developers who make the leap to develop a product are another story. Building a product involves a *large up-front time investment*, and as a result is far riskier than becoming a consultant because you have to wait months to find out if your effort will generate revenue. In addition, growing a product to the point of providing substantial income is a long, arduous road.

But let's say, for the sake of argument, that you spend 6 months of your spare time and you now own a web-based car key locator that sells 100 copies per month at $25 a pop. At long last, after months of working nights and weekends, spending every waking moment poring over your code, marketing, selling, and burning the midnight oil, you're living the dream of a MicroISV.

Except for one thing.

The Inmates are Running the Asylum

In our completely un-contrived scenario you're now making $2500/month from your product, but since you make $60k as a salaried developer you're not going to move back in with your parents so you can quit your day job. So you work 8-10 hours during the day writing code for someone else, and come home each night to a slow but steady stream of support emails. And the worst part is that if you've built your software right the majority of the issues will not be problems with your product, but degraded OS installations, crazy configurations, a customer who doesn't know how to double-click, etc…

The next step is to figure out, between the 5-10 hours per week you're spending on support, and the 40-50 hours per week you spend at work, how you're going to find time to add new features. And the kicker is that support burden actually worsens with time because your customer base grows. After 1 month you have 100 customers with potential problems, after a year, 1,200.

And yes, the person you decided to sell to even though they complained about the high price ($25) and then couldn't get it installed on their Win95 machine so you spent 3 hours on the phone with them and finally got it working only through an act of ritual

sacrifice is still hanging around, emailing you weekly wondering when the next release is coming out with his feature requests included (requests that not a single one of your other 1199 customers have conceived of).

But you persevere, and manage to slog your way through the incoming support requests and get started on new features.

What you find is that ongoing development, as with any legacy system, is much slower than green field development. You're now tied to legacy code and design decisions, and you soon realize this isn't what you signed up for when you had that brilliant flash of insight that people need web-based help locating their keys.

It's about this time that support emails start going unanswered, releases stop, and the product withers on the vine. It may wind up for sale on*SitePoint*, or it may be relegated to the bone yard of failed software products.

The Upside

The flip side to all of this is what you've already heard on the blogs of successful product developers.

Once a product hits critical mass you've conquered the hardest part of the equation. After that the *exponential leverage of software products*kicks in and you can live large on your empire of web-based unlocking-device locator applications. It's a recurring revenue stream that can grow far beyond what you would make as a consultant, all the while creating*balance sheet* value meaning one day you can sell it for stacks of proverbial cash and retire.

This is unlike your consultant buddy, whose consulting firm is worth about 42 cents (he had an unused stamp on his desk) once he decides to retire.

But there is a *dip* before you get to this place of exponential leverage and proverbial cash. A big dip. And if you can get through it once, it's more likely that you'll be able to get through it with your next product. And the one after that.

Once you make it to the other side, you've learned what it takes to launch and maintain a product, and next time you will have a monumentally better chance of success because you are now a more savvy software entrepreneur.

Congratulations! Go buy yourself a nice bottle of wine and sit back, relax…and enjoy answering your support emails.

Original Location

http://www.softwarebyrob.com/2008/11/18/the-software-product-myth/

Running Your Company

From 160 Hours to 10: A Tale of Agile Business Practices

More than two years ago my business partner and I discussed launching a hosted version of my ASP.NET *invoicing software*, DotNetInvoice.

We developed the plan and a task list, and estimated the effort at around 160 hours including development time needed to make DotNetInvoice a multi-tenant application. But given the heavy competition in the hosted invoicing software market and the level of effort of the task, it was continually placed on the back burner.

Until we figured out how to get to the same endpoint with 10 hours of work.

The Shearing

After our initial 160 hour estimate, every six months or so for the past two years we've revisited the idea of a hosted version until one day in November of last year. On this day we came to the realization that we didn't need to make DotNetInvoice multi-tenant in order to have a hosted version. The other invoicing players are multi-tenant because they coded their invoicing services from scratch, but it's not the only way to do it.

The more we thought about it we realized the benefits to keeping it a single-tenant application: the ease of migrating from our hosted version to a downloaded version, and keeping each customer's data separated in its own database to name two.

The interesting thing is that once that large task was removed from the list, other things to started to fall off, as well. At this point we begin looking at the launch of Hosted DotNetInvoice as a market test; to see if we could build enough of a customer base to warrant a major investment in the hosted invoicing market space. With that in mind, things started flying off our "must-have" list.

The next largest piece was automating sign-up and provisioning of a new hosted installation. In an ideal world, when a customer wants a new hosted account they would fill out a web form with all of their information and their new hosted version would be ready in 30 seconds. But that amount of automation – given the fact that we have to create a new sub-domain, a new database, and copy physical files – would take a substantial amount of time to develop and QA. So we tossed it.

The final thing we threw out was the need for a custom purchase page. A page where someone enters their details to make the purchase. In a desperate attempt to bring this entire project down to less than two days work we simply utilized PayPal subscriptions. Not the most professional approach, but it works very well for testing out the idea before we invest another day into this project.

Iteration vs. Automation

Now as a developer, the features I mentioned above seem like a necessity from day. *Not* automating this process creates the ongoing repetitive work that computers are designed to handle. Aaaaah, manual work…this is what computers are supposed to save us from!

In essence, this approach does not scale infinitely, which tends to be the kind of scaling that developers default to (this includes me).

But by getting over the need to automate everything to infinite scale and putting myself in charge of manually creating new hosted accounts, the time investment to get this feature launched dropped from 160 hours of work to about 10 hours.

I can hear the cries of developers around the world as I write this: "You can't launch a half-baked solution! You'll never go back and fix it!"

Most of us have worked in corporate environments where you're never allowed to go back and refactor code. This burns into our psyche that you don't want to launch a semi-functioning solution because you'll never have time to go back and fix it.

But the benefits of being my own boss and being a tiny software company, are that I can come back to this anytime. In fact, the more times I do it manually (since I'm handling it myself), the more motivation I will have to automate it. Add to that, the more I develop and streamline my manual process, the easier it is to automate it later using code. Having more experience with the process will actually mean I can code it faster.

And ideally, by the time I code it up, we'll have many customers using the platform which means I'll be working on a product I know is viable, and that's paying for the time I'm spending to automate it.

Agile Development, meet Agile Business.

Know Where it Ends

Through a bit of manual labor, or better yet some delegation and outsourcing, you can get to market with less up-front expense and in dramatically less time than if you try to automate everything.

But the number one question you need to think about when going after this Agile Business approach:

At what point is it worth spending the time to truly automate this?

In other words: how many sales do you need to make in order to justify the time and effort to write the code to automate this process?

Without a number in mind you run the risk of never wanting to invest the time to automate, and becoming hostage to a bunch of half-baked manual processes. At some point you have to automate the process or kill the feature…know that point before you start.

The Best Code I Ever Wrote

Had we chosen to automate everything, the worst outcome would have been investing 160 hours of time (as a *Micropreneur* that's a *huge amount of time*), and then scrapping the whole thing. When you're working on such a small scale you can't afford to throw away that much time.

But whether you're working as a corporate developer or a Micropreneur:

The best code you've ever written is the code you never had to write

Original Location

http://www.softwarebyrob.com/2010/04/07/from-160-hours-to-10-hours-a-tale-of-agile-business-practices/

What the Beatles Can Teach Us About Starting a Company

Photo by *John McNab*

Popular musicians possess entrepreneurial qualities, most often the passion and drive we see in successful startup entrepreneurs.

What we don't often think about is that although there's luck involved in becoming a "rock star," there are deliberate decisions a musician makes early in a career that will impact their level of success months or years later.

And so it is with starting a company.

While this article is meant as a break from the normal seriousness of this blog, I honestly believe that we can learn much from a group of founders who sold *hundreds of millions* (*some say 1 billion*) units.

Here are four lessons we can learn from the Beatles' startup career.

Do It for the Love, Because the Money's a Long Way Off

John Lennon and Paul McCartney, the two founding members of the band, would regularly cut school to write songs together. When kids cut school in my hometown, they went to play video games or smoke. Heading out from school to hone your songwriting skill is not on the radar for most high schoolers.

But they did it because they *loved writing music*. If they had done it for the money they would have stopped long before they became an overnight success five years of non-stop practice later.

This reminds me of a quote I've heard attributed to Stephen King, who announced his retirement a few years ago (though he did not actually retire). He was asked, "So, you're going to retire from writing?"

He replied, "No, I'm going to retire from publishing."

Those people who just *have* to do what it is they are doing, whether it's writing songs or writing novels, have a huge advantage over those who are in it for other reasons (typically money and fame).

I'm not a believer that you *have* to love the startup idea you're working on, but you do need to love the process of starting up…the excitement of breathing life into a new idea. Otherwise you're going to close up shop at the first sign of a *danger point*.

Get Feedback Early and Often

The Beatles began playing out under various names within a few months of forming in 1957, when John and Paul were 16 and 15, respectively. Though the line-up was ever changing and they didn't find a permanent drummer until 1960.

But although the band was still very much an unfinished prototype of what they would become, with a revolving door of musicians and numerous band names (including "Johnny and the Moondogs" and "The Silver Beatles") they were playing in front of audiences whenever possible.

This improved their musical abilities, was an instant feedback loop on which of their songs were "hits" and which were duds, and dramatically improved their stage presence. Each element was a piece that later fit together to build the best-selling musical group of all time.

So get out of the basement and talk to potential customers. Think of it as improving your stage presence.

Put in Your Time

During the two years from 1960 to 1962 The Beatles played countless all-night sets at clubs in Hamburg, Germany. These were eight-hour sets, and they would reportedly take uppers to maintain their energy on stage. Not only was it grueling work, but the sheer volume of material they had to learn to play eight hours worth of music is mind-boggling.

But this was how they built their reputation as a solid, reliable band with outstanding musical chops. They hung around other musicians and infiltrated the music scene in Hamburg and their native Liverpool, England and began seeing opportunities emerge, including backing a well-known singer of the day named Tony Sheridan on a version of *My Bonnie*.

This didn't happen by accident. As I've said before:

The more you work the "luckier" you seem to get.

I have yet to see a startup succeed without countless hours of hard work. You've heard this before so I won't belabor the point.

Build a Fan Base, then Go Big

The Beatles did not start touring Europe a week after they formed. Instead they played back and forth between Liverpool, England and Hamburg, Germany for two years building a fan base. Finding people who liked their music. Or their look. Or their style.

Before they went big they honed their writing abilities, musicianship, and their stage act. Only after two years and hundreds of hours of playing in front of a slowly growing group of fans did they "launch" by recording an album and heading out on tour.

Original Location

http://www.softwarebyrob.com/2010/08/25/what-the-beatles-can-teach-us-about-starting-a-company/

Build your fan base. Find those people who need your product. If you can't find this group of people, you're probably building the wrong product.

How to Detect a Toxic Customer

Photo by *Francisco*

A month ago I received a sales inquiry via email for my *invoicing software*package. The prospect asked if we could complete the questions he had attached in a spreadsheet:

> *I will need the attached questions answered in order to proceed as I can't get them all answered off your website.*

There were nearly 80 questions, at least half of which could be answered from our website.

In addition, he mentioned doing a flat-file exchange of data between our software and a custom piece his colleague had written. I mentioned that we have a .NET API or a web service layer, and that passing flat files back and forth would not be an optimal approach for a few reasons.

And that's when it started to get good.

A snippet from his disgruntled reply:

> *My colleague is a computer engineer and has been programming for over 25 years so he knows what he's doing. I just need pricing and questions answered at this point, thanks.*

Wow, nice guy so far! I wasn't trying to bust his chops, just letting him know about a potentially better approach.

Toxicity

After a few more emails back and forth it became obvious that not only might our system not be a good fit, but this prospect was very well not a good fit for our company.

It was apparent to me that even if this person bought our software after what looked to be an extensive due diligence likely to take up several hours, that our trouble would be just getting started.

Few things are worse than supporting a demanding, entitled customer who feels that their purchase price buys them control over your life, liberty and pursuit of happiness. I've run into my fair share over the past several years, and I call them *Toxic Customers*.

I've had toxic customers pull all of the following:

- Call my cell phone five times in 5 hours on Memorial Day (A U.S. Holiday). The customer was U.S.-based and aware of the holiday.
- Demand (not just ask for) support outside of our standard support hours because he was busy at his day job during our normal hours.
- Consume 10 hours of support time over 2 months because he did not follow the installation instructions properly, and did not perform the troubleshooting steps I sent in my first response to his ticket. 30+ emails later I logged into his system, performed one check and found the issue.
- And on, and on…

And while you (luckily) won't encounter many toxic customers during your lifetime, after the first few you learn how to identify and gracefully step away when you see them coming. This is because toxic customers are not just a hassle, they can chew up support time, cost you money, damage your reputation by posting to Twitter/forums/review sites, and stress you to the point of wanting to commit an act of violence on yourself or others.

Early Detection is Key

After dealing with several toxies over the past few years I've begun to watch for warning signs during the sales process, and have become more adept at identifying potentially harmful customers early on in the process.

Any one of these warning signs is not a big deal, but stack 2 or 3 on top of each other and (depending on their severity) you have yourself a red flag.

Warning Sign #1: Disrespectful or Abrupt

People sometimes act as if they are not emailing a real person, and that's ok. Sometimes we'll receive an apology after we reply with a respectful answer.

The person might be in a hurry, they might not know email etiquette, or they might be a jerk. It takes several emails to figure it out.

Warning Sign #2: Asks for a Discount (With No Reason)

Perhaps the second most common red flag is someone who asks for a discount with no real justification. We always work with particular circumstances, especially schools and non-profits. But asking us to drop our price by 25% or 30% just for kicks is not typically a sign of an outstanding customer.

Warning Sign #3: Multiple Contacts, Often Through Multiple Channels

One of my favorites is to receive 3 emails (one each to our sales, support and info addresses), and a voice mail from the same person within a few minutes.

The request is not time-sensitive, the person just want to make sure someone receives it. And we do receive it…four times.

I've never really understood this one; either they are really anxious to have their question answered, or they don't believe they are going to hear back. Either way, if you have to send four requests in order to hear back do you really want to buy software from that company?

Warning Sign #4: Unrealistic Expectations

Another good one is receiving 4 emails in two hours with escalating urgency, all surrounding a non-time sensitive pre-sales question that can typically be answered on our website. Something like:

Is your software localized for Australia?

10 minutes later:

I wanted to make sure you received the previous email. Is your software localized for Australia?

30 minutes later:

Hello, is anyone there? I haven't heard back from my previous email. Is your software *localized for Australia?*

20 minutes later:

WHY AREN'T YOU ANSWERING? DOES YOUR SOFTWARE HANDLE AUSTRALIAN DOLLARS OR NOT?!

Warning Sign #5: Multiple Questions that Can Be Answered from Your Website

This one is common, and far from a deal-breaker. But it could be a warning sign of a future support burden, especially if you have an installation process that requires them to read and perform a list of steps.

Now Back to the Story…

With the next step of filling out the 80 question spreadsheet staring me in the face, I opted to let the person know that our company was not a good fit for their needs. I wrapped up our conversation with:

> *Thanks very much for your interest. I think you will be better off finding another invoicing solution.*
>
> Turns out we were just getting started. A few hours later I received the following (snipped for brevity):
>
> *Please explain why you are recommending I find another product. I don't appreciate being sent off without an explanation. I will make that decision, not you. If it won't work tell me why, don't just tell me to find another product.*
>
> *If you don't want to work with me that's fine, just forward me to another sales rep, but don't shoe me off. I'm trying to conduct business here, I don't have time for games. One last thing Rob, why don't you use a last name?*

Bingo! I especially liked the comment about my last name.

Yes, I happened to sign an email without my last name. Definitely a sign that I am hiding something, and that my company and/or software is woefully deficient in many ways.

I wrapped up our conversation by respectfully pointing out that at our price point we are unable to handle a manual sales process where we answer a large number of questions that can be answered from our website. I also mentioned the following:

> *I'm confident that in the end, if you purchase, your approach of using test file between applications is going to turn into a support headache. We don't recommend that approach and are not willing to support it.*
>
> *I genuinely think you will not be happy with our application and will be better off with a different solution. We reserve the right to sell our software (or not) to whom we choose, and I do not see this as a good fit for either party.*
>
> *So once again, thanks for your interest. Best of luck finding a solution that fits your needs.*

I had done my best to make this interaction as respectful as possible, without watering down the message. The bottom line is that I was pretty sure that this was not someone we wanted as a customer (and was willing to wager a sale on it). I'd been down this path before.

The next morning I received the following email from someone at the same company:

> *My colleague is the wrong person to be heading up this project and I apologize for his monster list. Can we hit the reset button on this please? Our needs are really pretty simple.*

He listed four bullet points, all of which we handle out of the box. We exchanged 2 or 3 emails and he purchased within a week.

If you've dealt with a toxic customer let's discuss it in the comments.

Original Location

http://www.softwarebyrob.com/2010/12/09/how-to-detect-a-toxic-customer/

Why You Should Build Your Startup to Sell (But Not to Flip)

Photo by *mrphancy*

I recently started listening to the *E-mything Your Business Podcast* which is essentially a nine-episode commercial for a book called *Built to Sell: Turn Your Business Into One You Can Sell*. But it's not a bad commercial; it has interesting information about the factors that can substantially increase the value of a business.

This ties in with something I've said before:

Build your business to sell, even if you have no plans to.

This is because the factors that make a business attractive to potential acquirers are the same ones that make your business a better business to own: efficiency, automation, repeatable, documented, scalable, etc…

Notice I'm not talking about building your business "to flip." Flipping implies building a shoddy company that maximizes short-term profit.

Instead, we're talking about building your startup to maximize its value based on factors the market values, since these factors will make it a better company to own in the both the short- and long-term.

A Product that Scales

The author, John Warrillow, says that most businesses do not scale well. By "scale" he means: can the business grow very quickly without a lot of manual intervention?

A consulting firm scales very poorly. You've likely tried consulting work or are still doing it, and you know it's a dollars-for-hours trade. Nothing scalable (or fun) about that. This is why you're thinking about a startup, right?

The three components to a scalable business, according to John, are being:

- Repeatable
- Teachable
- Valuable

Let's take a look at each one of these in the context of a startup.

Repeatable

In a traditional business a repeatable product/service is something that does not require a high level of expertise and customization every time you make a sale. The sales process might be customized, but the product or service should be as cookie-cutter as you can make it. The more repeatable, the more profit you will make with each sale, and the more valuable your company becomes.

In a startup, this means developing a product and a sales process that does not require your time. Ideally you want the entire sales process to take place on your website with no manual intervention, including the purchase, order fulfillment, serial number generation, etc…

The ability to leave your startup on auto-pilot for a few weeks while you're out of town (or working hard on new features) means more profit per hour for you, and a higher valuation should you ever decide to sell.

The lesson: build a product that does not require customizations, and completely automate your order fulfillment process.

Teachable

In a traditional business you leverage the time of your employees. You pay them less than you bill out, and you keep the difference. This means that the easier your process is to teach, the easier it is to find employees who can do the work, the more employees you can hire, and the more profit you will make per sale.

In a startup you have a few avenues: hire employees or use virtual assistants (VAs).

In either case it's the same: having a process in place to handle customer support, refunds, product fulfillment, documentation updates, forum support, website updates, etc. will result in an easier ramp-up period for new hires , and less of your time that's wasted in training.

The lesson: create a written for everything in your business, even if you're the only person doing the work.

Valuable

John talks about having a product that's valuable to your customers. In a traditional business this means offering non-commodity products and services.

For a startup, this means avoiding "me too" products, staying away from horizontal niches, solving a pain-point, and targeting a vertical niche.

By catering to a specific audience (plumbers, grocery store owners, psychologists, etc…) and building a product (and marketing) that caters to their every need…you will build something far more valuable than if you built the same product for a massive, horizontal audience.

The lesson: stick to vertical niches. Occupations and hobbies are best.

Charge Up Front

The other point I've taken away from the podcast is the fact that businesses that are the most valuable are able to charge for their work up-front. So instead of selling a product or service and invoicing in 30 or 60 days, the most valuable businesses charge before the product or service is supplier.

This allows for better cash-flow, potential profitability from day 1, and no need to take out a line of credit.

Applying this to startups is a no-brainer; most of us will charge the customer's credit card before we provide a product.

The real application (and perhaps the punchline) is to charge for your product.

There are arguments against charging for your product, but unless you have venture funding they do not apply to you.

Original Location

http://www.softwarebyrob.com/2010/07/01/why-you-should-build-your-startup-to-sell-but-not-to-flip/

Recruiting & Retaining Developers

How to Recruit a Developer Entrepreneur for Your Startup

Lately I've spent a lot of time thinking about internet startups and entrepreneurial software developers (or as I like to call them, "developer entrepreneurs").

And lo and behold, no sooner did I sit down to write about it than I received an email with my "prompt" for this post. It went something like this:

> *For several years I have been refining a business idea to be conceived as an Internet startup. I have met with numerous developers over time and even after they express support for the business concept, my challenge has been in persuading web developers in partnering in the opportunity as entrepreneurial venture.*
>
> *Could you suggest to me an approach I could take to persuade Developers, to see such opportunities as business ventures instead of a project/job?*

> *How should I go about selling the business concept enough to have development of the application without initial funds?*

If you're a non-technical founder looking for a developer entrepreneur, these are questions you should ask yourself. Having been on the developer side of the coin a number of times, here is my take.

Code, Marketing and Money – Pick Two

If you take nothing else away from this post, remember this:

> *There are three components to bringing a web startup to market: code, marketing and money. You need at least two of them to succeed.*

Often, if you have one of the three you can find someone willing to partner with you.

If you have coding skills, find someone with marketing skills who's willing to become a partner. If you have money, pay someone to do the marketing.

If you have marketing skills, find a developer who's willing to become a partner. If you have money, pay someone to do the development.

If you don't have coding or marketing skills but Uncle Buck just left you a mad stack of cash, put together a good team, give them a truckload of Benjamins, and send them on their way.

Before you approach a potential partner, figure out which of the three you bring to the table. If you don't have any of them, re-consider the idea of an internet startup (or any software startup). I say that because *a good idea does not count as one of the three*.

You're looking for an experienced, professional software developer, right? If you're not an experienced, professional marketer, then why would a developer devote hundreds of hours of spare time to your project?

To summarize: if you are a non-developer looking to woo a developer, the first and most fundamental asset you need is experience bringing a product to market, or money.

Buy In

Since most people reading this article will not have said truckload of Benjamins, let's assume you have marketing experience. In that case, how should you go about courting a developer for your risky, unfunded startup? In much the same way you would court an investor: *show them that the idea will succeed*.

Show them a business plan (even if it's just a long email), research, numbers, specs, sketches, designs. Show them you have done your research and are dedicated to the idea.

You're trying to convince an investor (not of money, but of time) to fund your company. Your idea has to sound so good that this developer, someone who sees things in black and white, can't help but volunteer hundreds of hours of their nights and weekends. Don't bring a three-sentence summary and a quote from Us Weekly as your market research.

If you have trouble convincing developers to partner with you then one of three things is happening: your idea isn't very good, you're not presenting it well, or you're presenting it to the wrong developers (see the next section). Once 3 or 4 programmers have heard your idea and walked away, start asking yourself what needs to change.

Where to Find Entrepreneurial Developers

Two bad places to look for entrepreneurial developers: your local financial institution's I.T. department, and Craigslist.

Enterprise I.T. developers make a lot of money, and they won't easily leave a six-figure job to do unpaid work. You'll wind up with a long development schedule since they can only work 10-15 hours per week, their #1 priority will always be their day job, their productivity is not very good in the evenings because they've been coding all day, and it's hard to depend on someone when their wife's knitting club gets in the way of a deadline. Moonlighting is more akin to hobby development, so think twice before going down this path.

Depending on your locale you might be able to find someone on Craigslist, but more likely you will find a beginning coder who won't have the ability or dedication to bring your product to market. It's worth a shot, but except in rare circumstances you'll be better off with the suggestion below.

After moving to New Haven I spent a few weeks searching online for local tech entrepreneurs (Craigslist and Google). I had no luck connecting with people and decided that no one in New Haven was starting tech companies.

When we recently moved across town due to some landlord issues, I happened to move into the house of the founder of a local tech company called Higher One. That was all it took: with this single connection I met with half a dozen entrepreneurs, angels, and developers in the course of 10 days (and many more since then).

The moral: figure out how to break into your local network.

Find a *local group* or search the web for "[your city] entrepreneurs." Look for tech-focused entrepreneur groups on Facebook or other social networking sites, but go beyond that and meet people face to face. Sure, this requires you to leave your house, be social, and invest your time, but the dedication of the people you meet will exceed those you find building flat-file importers at IndyMac.

How to Attract Software Developers that Fit Your Company

I've worked for startups and stalwarts, small shops and large corporations, firms where software is the means to an end, and firms where it was an end in itself. After years of exploring what I love about building software I've realized that coding for a 17-person dot com is a far cry from building enterprise software for a 300-person credit card company. It only took me eight years to figure out why.

There are a myriad of articles on interviewing, evaluating, and hiring software developers, but a topic that's rarely discussed is *how to attract software developers that fit your company's environment*. With the current state of the software job market,

it's critical as a hirer that you determine not only what type of developers will be happy in your environment (your "developer demographic"), but how you can become more attractive to that particular slice of the market.

Your Developer Demographic

As an example, IndyMac and Countrywide are large financial institutions with development offices in Los Angeles. They attract people looking for stable jobs with good benefits where they can code 9 to 5 and then go home and not think about it from 5 to 9. I envy people who do not grow bored with this environment; I really do. Life would be much, much simpler if I could handle it.

There are also tech startups that attract caffeine-junkies. Long hours with potentially large reward, and the prospect of building something really cool in a short amount of time that could make a difference in peoples' lives. Or, more likely, be relegated to the failed startup scrap heap. These companies should be looking for people who love using new technology, are typically younger, willing to take risks, and willing to shoot from the hip.

These may be extreme cases, but they illustrate the point: certain attitudes and personality traits play nicely with certain environments. Often, a developer who thrives in one environment will find that she slowly withers away in others.

The Three Dimensions

To get more specific, there are three dimensions to a company that most affect the internal environment for a software developer. It's critical that you know which of these your company falls into, and not only market to, but ensure you can retain developers who fit that demographic. The descriptions of the software developers who like to work at each corporate classification are generalized, but they serve as a guide to get you thinking about the personality of your ideal candidate.

Your company may not match exactly with one of the choices below each dimension, but do your best to categorize it. Many companies start off as one thing and transition to something very different in the first year or two.

1. **Size**

   - **Small** – Some small companies are startups, and some are 10-year old, profitable, mature businesses that make money hand over fist with 9 employees. Software developers who like small companies are typically social, they like the vibe of knowing everyone, going to lunch with the same people, and knowing that they contribute a great deal to the success or failure of the enterprise.
   - **Large** – The bulk of corporate development is for larger companies. They tend to have big teams, lots of process, and decent-sized QA and Change Management teams. Software developers who like large companies tend to enjoy more process, like working on larger teams where they can either lead or be led, and enjoy the possibility for growth that comes with a large organization.

2. **Chaos Level**

   - **Stable** – Stable companies tend to be, um…stable. They have good benefits, and employees can often get away with working 8 or 9 hour days. Developers who prefer stable companies are likely a little further along in their career, may have a family, like the consistency of coming in to the same office each day, and enjoy their time out of the office when they don't have to think about software.
   - **Startup** – Startups tend to be more risky with the possibility of more reward. The salary may not be as high as that of a stable company, but the stock options will be worth six-figures if you can get your bleeding-edge online calendar out the door before Google's. Benefits tend to be spotty. The hours are long, but it's more than worth it for developers who are passionate about technology (read: bring books about ASP.NET to the beach), love building software that matters, and enjoy the camaraderie of working on a team of people who share their interests.

3. Focus

- **Software (or a technology that relies on software)** – These companies rely on software for their main source of revenue, whether they sell it (Microsoft, Intuit), give it away (Google, Craigslist), or offer it on some type of "pay to play" basis (eBay, salesforce.com). Technology firms that rely on software are companies like Palm, Apple, or the guys who make the fingerprint readers I keep seeing at data centers. Software may not be their main source of income, but they are technology companies and software is critical to their product. Software developers who prefer these types of companies enjoy being around other software people; they like the idea that technology is at the core of their company, and they love that they work on real products that people use. In addition, they relish the fact that software firms tend to live towards the cutting edge and are able to constantly upgrade their skills to the latest and greatest.
- **Everything Else** – These companies make up the majority of companies in the world; their main source of revenue is something other than software – credit cards, lawn furniture – you name it. "Everything else" companies use software to support their business: to track widgets, support their call center, and balance their books. The majority of corporate software jobs are working for companies under this umbrella. Software developers who enjoy this type of company like building applications to support accounting, management, and the people who produce or sell the company's main product, and they are either very content to know they can go home at night and not think about developing software, or they go home at night and all they think about is they day they can leave the company and work for someone like Google.

Your Company, Your Developer Demographic, Your Job Description

Before writing your job description think hard about where you fall in each of the dimensions. Then think twice as hard about the kind of developer who will be happy

at your company. Don't kid yourself; you can convince the 24 year-old tech hot-shot to work for your financial services company by offering him a huge salary, but he'll be gone in 9 months because you're using three-year old technology and developing boring (from his perspective), back-office software.

Once you've determined who will be happy at your company, *write your job description with that person in mind*.

You're a large company with great benefits? Spell it out in bullet points: plain, simple, and official. *Play up your stability*.

You're a small startup with no benefits? Use a casual tone and create excitement. *Make that 24-year old hot-shot be dying to work for you*.

Even if he has to pay for his own health care.

Original Location

http://www.softwarebyrob.com/2008/03/13/how-to-recruit-a-developer-entrepreneur-for-your-startup/

Personality Traits of the Best Software Developers

I come from the world of corporate software development. It may not be the most glamorous side of software (it's nowhere near as interesting *as shrinkwrap startups* or those fancy-dancy *Web 2.0 companies* that show up in your browser every time you mistype a domain name), but it's stable, pays well, and has its own set of challenges that other types of software development know nothing about.

For example, when was the last time someone working on the next version of Halo spent three weeks trying to gather people from accounting, marketing, product management, and their call center in order to nail down requirements that would likely change in 2 months once they've delivered the software?

Or when was the last time someone at *37Signals* sat through back to back weeks of *JAD sessions*?

In this world of corporate development I've known a few phenomenal developers. I'm talking about those A+ people whom you would quit your job for to go start a company. And the the more I looked at what makes them so good, the more I realized they all share a handful of personality traits. Well, not *exactly* a handful, more like four.

Pessimistic

Admiral Jim Stockdale was the highest ranking US military officer imprisoned in Vietnam. He was held in the "Hanoi Hilton" and repeatedly tortured over 8 years. Stockdale told Jim Collins, author of *Good to Great*, "You must never confuse faith that you will prevail in the end, which you can never afford to lose, with the discipline to confront the most brutal facts of your current reality, whatever they might be."

After his release, Stockdale became the first three-star officer in the history of the navy to wear both aviator wings and the Congressional Medal of Honor.

Stockdale was a pessimist in the short-term because he faced the brutal facts of his reality, but was an optimist in the long-term because of his confidence that he would prevail in the end.

No one anticipates a catastrophic system failure by looking on the bright side. The best developers I know are experts at finding points of failure. You'll often hear them quipping "What could possibly go wrong?" after someone makes a suggestion to handle a critical data transfer via nightly FTP over a dial-up connection. The best developers anticipate headaches that other developers never think of, and do everything within their power to avoid them.

On the flip side, great developers are optimistic, even downright confident, about their overall success. They know that by being a pessimist in the short-term, their long-term success is ensured. Just like Jim Stockdale, they realize that by confronting the brutal facts of their current reality they will prevail in the end.

Angered By Sloppy Code

Paul Graham nailed it when *he said* "…people who are great at something are not so much convinced of their own greatness as mystified at why everyone else seems so incompetent."

The worst nightmare for a great developer is to see someone else'ssoftware gasping for air while bringing the rest of the system to its knees. It's downright infuriating. And this isn't limited to code; it can be bad installation packages, sloppy deployments, or a misspelled column name.

Due to the life and death nature of their products, NASA designs zero-defect software systems using a process that has nearly eliminated the possibility for human error. They've added layer after layer of checks and balances that have resulted from years of finding mistakes and figuring out the best way to eliminate them. NASA is the poster child for discovering the *source* of a mistake and modifying their process to eliminate the possibility of that mistake ever happening again. And it works. A quote from *this Fast Company article* on NASA's development process says

"What makes it remarkable is how well the software works. This software never crashes. It never needs to be re-booted. This software is bug-free. It is perfect, as perfect as human beings have achieved. Consider these stats: the last three versions of the program — each 420,000 lines long-had just one error each. The last 11 versions of this software had a total of 17 errors. Commercial programs of equivalent complexity would have 5,000 errors."

I'm not saying we have to develop to this standard, but NASA knows how to find and fix bugs, and the way they do it is to find the source of every problem.

Someone who fixes a problem but doesn't take the time to find out what caused it is doomed to never become an expert in their field. Experience is not years on the job, it's learning to recognize a problem before it occurs, which can only be done by knowing what causes it in the first place.

Developers who don't take the time to find the source often create sloppy solutions. For hundreds of examples of sloppy solutions visit *The Daily WTF*. Here are a few I've seen in my career:

- An assembly is deleted from a server each time it's rebooted. You could create a custom script to re-copy that assembly to the server after each reboot, or find out why the assembly is being deleted in the first place.

- An image-manipulation script is hogging processor power for minutes at a time when it should run in under 10 seconds. You could make the script run at 2am when no one will notice, or you can take the time to step through the code and figure out where the real problem is.
- A shared XML file is being locked by a process, causing other processes to fail when they try to open it. You could make several copies of the XML file so each process has its own, or you could troubleshoot the file access code to find out why it's locking the file.
- And on and on…

Long Term Life Planners

This one was a little puzzling for the longest time, but I think I've finally put it together.

People who think many years down the road in their personal life have the gift to think down the road during development. Being able to see the impacts of present-day decisions is paramount to building great software. The best developers I know have stable family lives, save for retirement, own their own home, and eat an apple a day (ok, maybe not that last one). People who have spastic home-lives and live paycheck to paycheck can certainly be good developers, but what they lack in life they tend to lack in the office: the ability to be disciplined, and to develop and adhere to a long-term plan.

Attention to Detail

I've known smart developers who don't pay attention to detail. The result is misspelled database columns, uncommented code, projects that aren't checked into source control, software that's not unit tested, unimplemented features, and so on. All of these can be easily dealt with if you're building a Google mash-up or a five page website. But in corporate development each of these screw-ups is a death knell.

So I'll say it very loud, but I promise I'll only say it once:

I have never, ever, ever seen a great software developer who does not have amazing attention to detail.

I worked with a programmer back in school who forced anyone working with him to indent using two spaces instead of tabs. If you gave him code that didn't use two spaces he would go through it line-by-line and replace your tabs with his spaces. While the value of tabs is not even a question, (I've long-chided him for this anal behavior) his attention to such a small detail has served him well in his many years designing chips at Intel.

So There You Have It

The next time you're interviewing a potential developer, determine if she has the four personality traits I've listed above. Here are a few methods I've found useful:

- Ask if they're an optimist or a pessimist
- Ask about a time when they found the source of a problem
- Find out if they save for retirement (you can work this in during discussions of your company's retirement plan)
- Make an obvious misspelling in a short code sample and ask if they see anything wrong

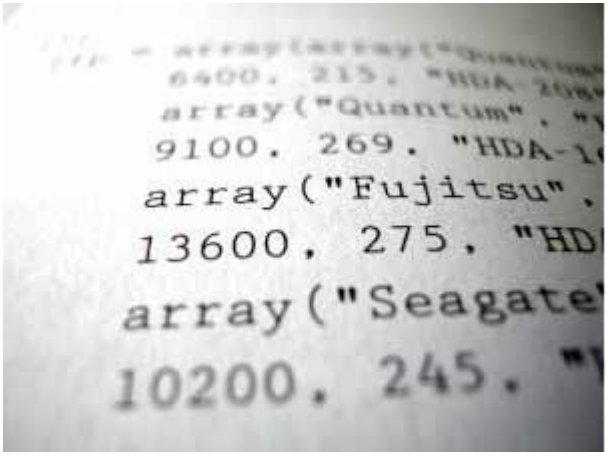
We know from *Facts and Fallacies of Software Engineering* that the best programmers are up to *28 times better* than the worst programmers, making them the best bargains in software. Take these four traits and go find a bargain (or better yet, make yourself into one).

If you liked this article you'll also like my article *Timeline and Risk: How to Piss off Your Software Developers*.

Original Location

http://www.softwarebyrob.com/2006/08/20/personality-traits-of-the-best-software-developers/

Nine Things Developers Want More Than Money



Many of the developers I know have been programming since they were in junior high. Whether it was building text-based games on an Apple IIe or creating a high school football roster app in Visual Basic, it's something they did for the challenge, for the love of learning new things and, oh yes, for the ladies. Ladies love a man who can speak BASIC to his Apple.

College graduates face a sad reality when they leave the protective womb of a university and have to get their first real job. Many of my friends found jobs paying around $25k

out of school, and were amazed that the starting engineering and computer science salaries were nearly double that. But the majority of the engineers in my class didn't become engineers for the money; we did it because it touched on a deep inner yearning to tinker and impress their friends. And did I mention the ladies?

Money is a motivating factor for most of us, but assuming comparable pay, what is it that makes some companies attract and retain developers while others churn through them like toilet paper?

Hygiene and Motivation

In the 1950s a researcher named *Frederick Herzberg* studied 200 engineers and accountants in the US. He asked them a few simple questions and came up with what is one of the most widely-accepted theories on job satisfaction called *Two Factor Theory*.

His theory breaks job satisfaction into two factors:

- *hygiene factors* such as working conditions, quality of supervision, salary, safety, and company policies
- *motivation factors* such as achievement, recognition, responsibility, the work itself, personal growth, and advancement

Hygiene factors are necessary to ensure employees don't become dissatisfied, but they don't contribute to higher levels of motivation. Motivation factors are what create motivation and job satisfaction by fulfilling a person's need for meaning and personal growth.

Think of a large financial company like Countrywide or IndyMac. Although I've never worked for either, the stories I've heard indicate the hygiene factors are well taken care of: working conditions are good, supervision is reasonable, salaries are decent, they have good benefits, etc…

However, the motivation factors are, shall we say, incognito. As Herzberg noticed, this scenario leads to employees viewing the job as little more than a paycheck, which is probably all right for companies like Countrywide and IndyMac.

Take the flip side: a tiny startup in a dingy office with no windows, crappy benefits, little supervision (because the CEO's on the road making sales), and no company policies (because the CEO's on the road making sales). But the constant rush of learning, being responsible for the company's success or failure (almost single-handedly at times), and believing in the company's future growth makes this job much more desirable for many developers.

One of my early programming jobs was for a web consulting startup during the dot-com boom. There were 7 of us (we grew to 17 during the height of the boom) shooting each other with water pistols, throwing Nerf footballs around the office, and cranking out insane amounts of caffeine-driven code. We learned a new language every project and were always on the cutting edge.

I remember thinking that a company across town could have offered me a $15,000 dollar raise and I wouldn't have taken it. The motivation factors were overpowering.

On the flip side, the benefits were terrible, the office was a series of tiny cubicles, gray from years of neglect – Smurf-blue network cables hung from the ceiling, and supervision was…well…non-existent. And although hygiene factors were lacking, developers flocked to work for this company and only one left while I was there.

She was interested in a more stable work environment and better benefits, and went to work for a large financial institution much like IndyMac.

Rob's Criteria for Keeping Your Developers Happy

If you want to collect a paycheck for 25 years and retire with a gold watch and a pension then go for companies that have the hygiene factors nailed. Stroll in at 8, head for the door at 4:59, and count the years until you're kicking up your feet on a beach bar in Costa Rica.

But if you're reading this, odds are that you aren't the kind of person who never thinks about code after 5:01; you're more likely to have a collection of DVDs that come up in an *Amazon search for "Silicon Valley*." You're probably one of those people who*needs* motivation factors or you go crazy with restlessness, and when the motivation factors are in place you'll work ridiculous hours for low pay just because it's so damn fun.

I talked to a dozen colleagues and pored over my own experiences to arrive at this list of nine software development motivation factors – *Rob's Criteria for Keeping Your Developers Happy*.

There's only one rule when determining your score: your vote doesn't count unless you're a developer. If you're not in the trenches writing code then forward this article to someone who does and ask for their opinion. In addition to keeping management from making an unfair assessment, my greater hope is that this *inspires conversation* and forces management and developers to talk about these issues so we can get them out in the open.

Without further ado, here they are:

1. Being Set Up to Succeed

It's a sad reality, but most software projects are set up to fail. Every developer has their horror stories; the "anti-patterns" of software project management. I've seen an architect given documentation for a legacy system that he pored over for week while designing a new interface for the product. After the design was complete he found out that the documentation was three years old and didn't reflect several major changes the system.

I've spent hours preparing a detailed technical estimate only to be told that the real deadline, already set by product development, gives me half the time I need.

Realistic deadlines are a huge part of being set up to succeed. Developers want to build software that not only works, but is maintainable; something they can take pride in. This is not in-line with product development's goals, which are for developers to build software that works, and nothing more.

The first thing to go when time is tight is quality and maintainability. Being forced to build crap is one of the worst things you can do to a craftsman. Delivering a project on-time but knowing it's a piece of crap feels a heck of a lot like failure to someone who takes pride in what they build.

It's critical to have buy-in to do things the right way, and not just the quick way. As one developer I talked to put it "Quality is as important as feature count and budget."

Schedule is not the only way a project can be set up to fail, but it is the most common. Others include: being forced to use cheap tools (be it software or hardware), working with a partner who doesn't deliver, bad project management (see #2, below), changing scope, and *unspoken expectations*, among others.

2. Having Excellent Management

Excellent management, both for projects and people, is a must-have motivation factor. This means no micro-managing, the encouragement of independent thinking, knowing what it takes to build quality software, quick decision making, and a willingness to take a bullet for the team when product development tries to shorten the schedule

These are the traits of an amazing software manager; the traits of a manager whose team would bathe in boiling oil to defend her, and work all-nighters to prove her right. When a manager takes bullets for the team, good developers tend to return the favor and then some. It creates an almost cult-ish loyalty, and the results are not only motivated developers, but insanely good software.

3. Learning New Things

Behavioral research indicates we're happiest when we're learning new skills or challenging old ones. A *recent article* cites a study by two University of Columbia

researchers suggesting that workers would be happy to forgo as much as a 20% raise if it meant a job with more variety or one that required more skill. This research suggests that we are willing to be paid less for work that's interesting, fun, and teaches us new skills.

This is why companies using Ruby can find experienced programmers willing to work for less than their typical salaries. The learning factor is huge when it comes to negotiating compensation.

Every developer I know loves playing with flashy new technologies. It was Perl and HTML in the mid-90s, ASP, PHP and Java in the late-90s, ASP.NET and XML a few years ago, and today it's AJAX and Ruby (and in some circles ASP.NET 2.0). Give someone a chance to use these toys and they'll not only be able to impress their friends, but fulfill that piece inside of them that needs to learn.

Keep a developer learning and they'll be happy working in a windowless basement eating stale food pushed through a slot in the door. And they'll never ask for a raise.

4. Exercising Creativity and Solving the Right Kind of Problems

Developers love a challenge. Without them we get bored, our minds wander, we balance our checkbook, check our email, hit Digg and Slashdot, read a few blogs, hit the water cooler, and see if any of our friends are online so we can once and for all settle the debate surrounding your uncle, the IDisposable interface, and that piece of toast shaped like the Virgin Mary.

I've watched developers on multiple occasions stay up until sunrise to solve a technical problem *without being asked and without extra pay*. The best developers are addicted to problem solving. Just drop a Sudoku in the middle of a group and watch them attack it.

Faced with the right type of challenge many developers will not stop until it's fixed, especially if it requires a particularly creative solution. Faced with the wrong type of challenge and they're back on instant messenger describing the toast.

The right type of challenge is a technical challenge that teaches a new skill, preferably one everyone's talking about. One example could be: "Consume these five RSS feeds, aggregate the data, and display the headlines on a web page…and figure out how to use AJAX to make it cool."

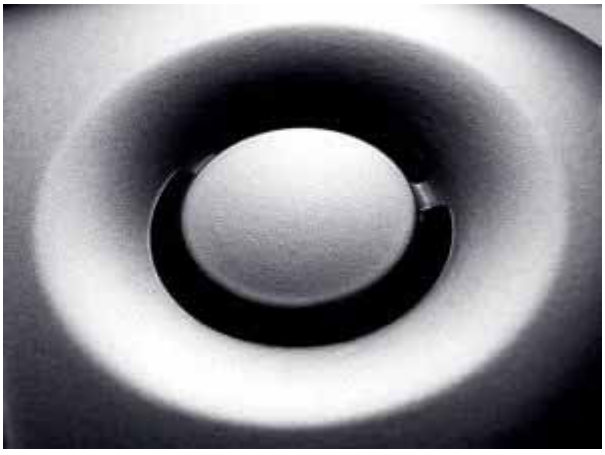
The wrong types of challenges are things like: "Fix that other guy's code. You know, the one we didn't fire because we were afraid he might cause problems. Well, he wrote a really crappy system and now we need to fix it and make it high-quality and maintainable. Oh, and you have until tomorrow."

If your business doesn't provide challenging work to developers, figure out how you can start. If there is no chance you'll ever be able to provide challenging work, find developers who are into hygiene factors, because developers who need motivation factors won't stay long.

5. Having a Voice

Developers are in the trenches, and they're the first ones to know when a system or process is not working. One developer I spoke with told me:

"[I want] someone to listen to my problems and actually take them seriously. I've worked at a few places where more RAM, more hard disk space, or faster/dual CPUs were simply not a priority for the company, but it was incredibly aggravating to the point of impeding my work. At one place I worked, every time I wanted to compile the software I had to clear all my temporary files because I needed more disk space. Talk about asinine. Being forced to work using outdated technology is really frustrating."



When a developer speaks, someone should listen. When several developers are saying the same thing, someone should listen and act…quickly.

6. Being Recognized for Hard Work

As engineers we love building things that impress ourselves and our friends. At least the ones who realize how hard it is to write a Perl compiler. From scratch. In FORTRAN. On a Vic 20.

Building something great is fun, but it's much more fun when someone's there to pat you on the back, throw you a party, sing your praises, or buy you a steak dinner. Most developers enjoy hearing praise and receiving recognition for hard work, but even the ones who don't need it are somehow soured when they don't receive it (or worse yet, someone else receives recognition for your work).

Recognition is one of Herzberg's core motivation factors and it applies to software developers as much as the engineers originally interviewed.

7. Building Something that Matters

Even though we're not medics in Bosnia or food carriers in Sudan, most people want to feel like we're somehow doing our part to make the world a better place,

both technologically and socially. Some of us might *think* we do it just for the sake of technology, but in the back of our minds we see ourselves as part of a grand scheme.

For instance, a friend of mine works for a financial company and cherishes every time they launch a product that helps the under-served financial community.

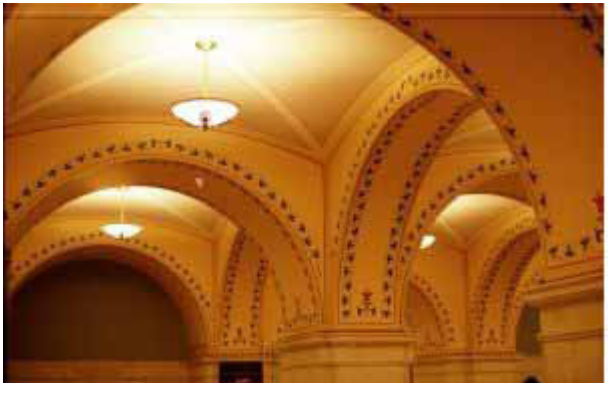
An Albertsons *inventory software* developer enjoys coming to work every day because his work ensures, via complex supply and demand algorithms, that the baby cereals are always available on the shelves.

Building something that matters makes an L.A. Times software engineer ecstatic that the trucks are now saving over 30% of their mileage and fuel costs due to his shortest path finding software implementation for newspaper delivery.

On the other hand, writing an interface to a buggy API that'll be used a total of 15 times in the next year doesn't seem like it matters much.

Copying and pasting an entire application and changing a bunch of labels isn't as exciting as it might sound.

And hacking in a few more case statements in a ridiculously complex stored procedure in order to service yet another customer without creating a proper data structure somehow doesn't seem to fulfill that part of us that wants to build something that matters.



8. Building Software without an Act of Congress

I was a contractor for three years starting in 2001, and during that time I built a ton of web applications. Since much of my development was off-site I became accustomed to writing software really quickly once we knew what to build. Another developer and I built insane amounts of software over the course of two years.

When I got my next full-time job it felt like I was dragging 50-pound weights. For every page I wanted to build I had to call a meeting with six people. Any change to the database required three approvals. It was nuts, and applications took 5x longer to build. Talk about frustrating.

The authority to make project decisions without calling a meeting is *huge*.

9. Having Few Legacy Constraints

No one likes developing against buggy interfaces, crappy code, and poorly-designed data models. Too many legacy constraints kill creativity, require an act of congress to modify, and generally sucks the fun out of building software (see several of the previous points for why this is bad).

If you have gobs of legacy liability, try to figure out a way to minimize its impact on future development. If you can't, look for people who value hygiene factors, because motivation factor developers are not going to maintain the same poor-quality applications for very long.

Determining Your Score

Let's face it, the bar has been set pretty low when it comes to motivating developers. How many companies can you think of that would score even as high as a 3?

Since this test hasn't been administered to companies across the globe there's no basis for comparison, but that's where you come in. I'd like to do an informal survey so we can get an idea of how things are in the real world. Please post your company's score in the comments (you don't have to post the company name).

Most large companies I can think of would be lucky to score a 1. Google would probably score an 8 or a 9.

Wrap Up

If you're a manager, when was the last time you asked your developers about these issues? If you're a developer, when was the last time you respectfully raised one of these issues, providing examples and a possible solution?

If the answer is "a long time ago" then you have some work to do. Send this article to a few of your colleagues and *start discussing how to enact change*.

Original Location

http://www.softwarebyrob.com/2006/10/31/nine-things-developers-want-more-than-money/

The Single Most Important Rule for Retaining Software Developers

There are good problems and bad problems when developing software.

A good problem is designing an elegant caching mechanism for your configuration variables, or determining how to architect your service oriented architecture so it doesn't look like the New York subway system.

A bad problem is figuring out how to re-architect your code to work with a partner whose API goes down twice a day, spending three hours trying to get your code into source control, or filling out ten minutes of paperwork for a five minute code change.

Paul Graham makes mention of this in *Taste for Makers*:

"Not every kind of hard is good. There is good pain and bad pain. You want the kind of pain you get from going running, not the kind you get from stepping on a nail. A difficult problem could be good for a designer, but a fickle client or unreliable materials would not be."

And in Great Hackers:

"One of the worst kinds of projects is writing an interface to a piece of software that's full of bugs…you don't learn anything from [doing this]. Writing a compiler is interesting because it teaches you what a compiler is. But writing an interface to a buggy piece of software doesn't teach you anything, because the bugs are random…Working on nasty little problems makes you stupid."

Bad problems are not interesting, not fun, and they teach you nothing. They create frustration and, given enough of them, will eventually cause burnout.

I've seen a string of bad problems last for months, and cause developer after developer to leave a company in search of more interesting work. Bad problems wreak havoc on your ability to meet *Rob's Criteria for Keeping Your Developers Happy*.

Look at any large financial institution, government agency, and [*insert name of large organization that considers developers to be a cog in their wheel here*]. I've worked at a few of these companies, and often hear developers who are out of work joking: "If things get bad enough I could always go work for Company X." Ouch…tell me that doesn't hurt your ability to find good people.

On the flip side, look at Google, Fog Creek Software, and SourceGear. I've never worked at these companies, but evidence is strong that good problems are at the forefront of their agendas. And by some shocking coincidence they get a bazillion resumes for every open position.

Recruiting & Retaining Developers

A steady stream of good problems is the foundation for keeping your developers happy. Minimizing bad problems and maximizing good problems should be every development manager's #1 goal.

Original Location

http://www.softwarebyrob.com/2006/11/17/single-important-rule-retaining-software-developers/

Micropreneurship

Ten Things I Will Never Have to Do Again

Almost four years ago, after a failed attempt to grow my consulting firm, I made the decision to never hire another employee.

Through my *experience as a manager* and employer I realized that the time and energy it takes to hire, manage, motivate, train, administer, and retain employees would be better invested in adding value to my products.

In other words, I'd rather spend my time writing code, talking to customers, and marketing my products than managing employees who would do that work for me.

As time has passed I've realized the benefits of Micropreneurship are abundant. Far more than I realized when I initially made the decision.

Some of the more obvious benefits include never spending a single minute worrying about:

- Employee retirement plans
- Payroll
- Hiring
- Firing
- Someone quitting
- Meetings
- The myriad of office politics I've experienced, even at small companies

And mark my words; every one of these takes time, energy and focus away from making your product better. More than you can measure.

But aside from these, there are some less obvious areas that we spend countless hours working on (and worrying about) that a single founder with no employees will never do again.

So with a sigh of relief I present to you a list of ten things that confirm Micropreneurship was indeed the right choice for me.

I will never again have to:

- Read a book about leadership
- Worry about building a company culture
- Get someone's "buy in"
- Participate in a performance review (on either side of the table)

- Motivate a hopeless employee that I did not hire and could not fire
- Negotiate a raise
- Commute
- Ask for permission to take a day off
- Take only two weeks of vacation in a year
- Update my resume

Would love to hear your additions to this list. Please post them to the comments.

Original Location

http://www.softwarebyrob.com/2010/09/29/ten-things-i-will-never-have-to-do-again/

Should You Build or Buy Your Micro-ISV?

Micro-ISVs. I've been contemplating the issue of building vs. buying for the past four years.

I've been on both sides of the coin: I've purchased 10 profit-oriented software products or websites, and built three.

Knowing what it takes to develop the initial version of a non-trivial software product (read: hundreds of hours), I've become a fan of buying. This is based on two factors:

- I have no spare time and a bit of spare money
- Hmm…no, I guess #1 is the only reason

As a software consultant I'm booked full-time and I bill a reasonable hourly rate. So to spend 348 hours (2 months) building a product means I'm approaching a mid-five figure investment into a software product. That's not play money; those are real dollars that don't wind up in my pocket.

And I don't have the confidence in my ability to know a market well enough that I would drop that kind of money on an untested product idea when there are less risky alternatives.

Looking at the products I've bought and built, none of them required skills beyond that of a mid-level developer. Sure, there are products that are more complex, but let's be honest, building an *invoicing system* does not involve insanely complex algorithms and coding chops. Most successful Micro-ISV products (and a lot of not-so-Micro-ISV products) could have been built by a few solid mid-level developers.

With this in mind, spending 348 hours of my time doesn't seem like the best business decision when I can:

- Hire someone to build the application (in my case, use the team I already have in place), or
- Find a proven product that may already have a customer base, sales website, etc… that I can buy for less than I can build it

You probably think I'm nuts, preaching "buy" over "build" to a group of software developers. So let's take a closer look at the scenarios:

Building It

I love writing software, so this has historically been my path of choice. However, the amount of money (based on lost consulting hours) I would spend on a 1.0, plus building a sales site, documentation, SEO, pay per click (PPC) campaign, etc… would be at least $40,000.

I have faith in my ability to build and market software, but that's a lot of faith to put into something that's generating zero cash. You'd be nuts to buy a software product with no revenue for $40,000.

However, if you want to run a Micro-ISV because you enjoy writing code, or you have a lot of non-billable spare time, then this is a viable option.

But I must caution you – laptops around the world are filled with the remnants of half-built products. Committing 200+ hours of your spare time to build and launch a product is no joke. Writing code 50 hours per week you would have a 200 hour project launched in 4 weeks…no problem!

But if you're coding in your spare time you'll be lucky to get in 10 hours of coding per week, and your productivity will be low because it will be 2 hour blocks when you're already tired from schlepping mindless reports all day for "the man." Trust me – I've done it. It's not easy.

Soon that 200 hour project turns into more than 20 weeks of your free time…almost 6 months. The first month is a breeze, it's the last five that'll kill ya!

Hiring It Out

Hiring someone to build your software is a good middle ground, and allows you to maintain some control over the technical piece without it sucking the coding life from your veins.

The advantage of hiring out product development is that it gives you time to build the sales site, write documentation, focus on SEO, marketing, PPC advertising set-up, payment processing, and the hundred other things I'm forgetting to mention.

If you're doing things right, the effort to get your product built is around 50% of the total time it takes to launch a Micro-ISV.

I've found success in outsourcing code and graphic design, and handling everything else myself. "Everything else" means the business side of things…the piece where you will likely learn the most, where you can bring the most value, and that you can't easily outsource.

And think about it…*a lot* of people can build a good invoicing application. A lot.

But how many can work the necessary marketing angles, form partnerships, create a profitable pay per click campaign, and build a compelling sales site? Finding someone

who can execute on these is much more difficult (and more expensive) than finding a developer who can build your application.

The single most important factor in the success of a Micro-ISV is marketing and sales, not the software itself.

In no way am I arguing for mediocrity in software development – your software has to get the job done. However, don't believe for a minute that great software beats great marketing. It never happens.

There's a reason *Bob Walsh* doesn't help developers write better applications. He helps educate them on sales and marketing.

FogBugz is good, but probably not the best bug tracking software on the market. Yet I bet it outsells most of its competitors by a huge margin based on marketing.

If you don't know how to work the marketing angles, form the partnerships, and do the other things I mentioned above you're going to need to:

- learn fast,
- find a partner, or
- stick to the day job.

Seriously…building (or buying) a great application is not going to get you there.

With this in mind, let's take a wild swing at the costs involved in this approach:

The graphic design and HTML will run from $500-$1500 if you offshore (optional depending on your personal view). Doing it in the U.S. will cost $2,000-6,000.

Two months of development (a safe estimate when hiring someone to build a small product from scratch) will run $14k-$21k here in the states, or around $7k if you offshore.

Total you're looking at $16k-$27k in the states, $8-9k if you offshore. These are obviously very rough numbers based on a typical Micro-ISV product requiring two months of development.

The potential pitfalls of this approach are obvious: if the developer is bad, you get software that doesn't work. A key strategy here is to screen your developer carefully and *only hire really good ones*.

Also, design the DB and screen mock-ups yourself. Not only will you get much closer to the product you envision, you'll be able to maintain it in the long-term.

Buying It

This is the approach I started favoring about two years ago. It started with my interest in buying (and later selling) domain names and websites. I soon realized that there are bargains to be had when buying a product or site that's already making money.

DotNetInvoice (my *ASP.NET billing* product) is a good example – I purchased the product, sales site, payment processing code, search engine rankings, and a small customer base for about 20% of what it would have taken me to build it, and yes, even cheaper than I could have hired someone to build it. It was built in Florida by two professional developers in their spare time. Quite a deal, indeed.

The reason these products and websites sell for such low valuations is that the market values revenue, and most of the product developers don't have the marketing and sales knowledge to bring their product to its full revenue potential.

This means there are completed software products and websites for sale, selling for literally pennies on the dollar compared to your cost to build them. I realize this sounds like a late night infomercial, but believe me, it's true. And how much would you expect to pay for this information? Just kidding…

The pitfalls of this approach:

- You're taking on risk in buying a product you didn't build
- You can't search for a specific type of product; for the most part you're limited to what's for sale

As an example, I didn't go looking for an invoicing system. I happened across DotNetInvoice and made an unsolicited offer. If you read my*original account of the*

purchase you'll know there were some early hurdles that I had to overcome. But once I worked out those kinks I've never doubted that I made the right decision.

One aspect I really like about buying a product is that it forces you, right off the bat, to not think about code.

As developers we want to spend all of our time working on technology because it's where we're most comfortable. But as I mentioned above the real hard work, and where you should spend the majority of your time, is on marketing, PPC, SEO, and partnerships. Buying a product forces you to think like this because the thing's already built.

If you don't want to spend the majority of your spare time on non-technical issues like marketing, I suggest partnering with someone who does, or sticking to the day job. The *day job will probably pay better*, anyway.

Original Location

http://www.softwarebyrob.com/2010/09/29/ten-things-i-will-never-have-to-do-again/

The Inside Story of a Small Software Acquisition (Part 1 of 3)

My startup history goes back a few years.

In 5th grade I sold comic books to my classmates at a 30-40% markup. I was a voracious marketer; I handed out homemade flyers, created checklists so customers could see "at-a-glance" which issues they needed, and even started a subscription service. Whenever kids in my school had extra money the first thing they thought of was buying comics.

In 8th grade I sold candy at a 500-800% markup because kids couldn't buy it within walking distance of school. I made money hand over fist, and quickly learned that you

should re-invest your profits instead of purchasing DJ equipment that you think will make you cool, but will actually collect dust in the back room of your house because you never spend the time to perfect your cross-fade.

In high school I wrote a booklet about comic collecting and sold it through classified ads. Technically I broke even, but realistically I lost money on the 50+ unpaid hours I spent researching and writing. This was the first business I launched "in the wild," and I learned a lot about what it takes to market a product in the real world (i.e., to someone other than my classmates).

During college I sold $5,000 worth of comic books on newsgroups and eBay (this was circa 1997, when eBay was still black and white and so slow you had to snipe 40 seconds before the auction ended or your bid wouldn't hit the servers in time). This business funded my entertainment expenses for two years. I had many *Silver Age* books that were some of the few copies for sale on the internet at the time.

Fresh out of college I started an ISP with my best friend. We closed down after a few months, but the technical experience we gained resulted in lucrative web development jobs for both of us.

Lessons learned so far? *Be the sole source, market like hell, go for big markup, and learn something from your business*.

These are basic business principles, but I learned them from hard-knox experience by the time I was 22.

Several years later I started my consulting firm, *The Numa Group*, which has been going strong in one form or another for five years.

In the past two years I've found time to build and launch *FeedShot*, Flogz (now defunct), and buy and sell an ASP.NET discussion forum package called *ChitChat.NET* (now owned by *Moon River Software*).

But the next entry in this saga, an ASP.NET *billing software* package, is quite a story. So let me start from the beginning.

Two Types of Leverage

There are two types of leverage: people and products[1].

A consulting firm ***leverages people***. The employees work for a lower rate than the company bills, and the owners keep the difference.

Leveraging people is lucrative if you can hire good people and keep them busy. There is little up-front risk as the company is (theoretically) paid by the client for most hours an employee works. The downside is that resource loading is difficult and you often wind up with people who are too busy, or not busy enough. Consulting firms are also notoriously hard to sell.

I was talking about leverage with *Joel Spolsky* at one of his recent appearances on the *FogBugz World Tour* (yes, that was a conspicuous name drop), and he nailed it: "The problem with consulting is that you can't find people who are as good as you." You may find a few, but the better they are, the more likely they are to go off on their own. So your growth, and thus your leverage, is limited by personnel (one of those human sides of software people keep talking about).

Now to product leverage: Microsoft, Oracle, and PeopleSoft are examples of companies that ***leverage products***. They invest in building a product once, and then sell it to many customers.

---

1 A third type of leverage isn't really leverage. Some people say you can leverage fame or popularity. Someone who writes books and articles can become a household names in a software community and command a high hourly rate for their consulting services. While it's true they can make substantially more money than their colleagues there is a limit to their earning potential, and that's why it's not truly leverage. Leverage scales. You can hire 10 or 1,000 consultants, or sell 10,000 or 100,000 copies of your software and earn egregious amounts of money without substantially more work. But no matter how much popularity someone achieves, their hourly rate will hit a cap, and they will still work 1 hour for x dollars. This is not a bad thing, mind you, it's just that it doesn't fit my definition of leverage.

Leveraging products is extremely lucrative if you can sell enough copies to make back your initial investment, knows as "sunk costs." Knowing how to market software, which I'm convinced is some kind of freaky black art, is crucial to generating enough sales.

One of the benefits of a product company is that it can be sold more easily than a consulting firm.

The downside is that the investment to bring a product to market can be sizable, and typically requires outside funding (or, for a smaller product, a lot of evenings and weekends), and that freaky black art of software marketing is harder than you think.

What's Next?

I come from the consulting world; I worked for small consulting firms on and off for five years. Climbing the ranks in consulting is obvious if you work for a large firm, but there's not much room to advance in a six person company. Seeking to learn more about the business side of things and to take the next step in my career, I started *The Numa Group* in 2002 and we are now a thriving three-person .NET development shop.

And in the midst of all this I've been looking for new skills to augment my consulting background; something new to keep the fire burning as brightly as in the early years of my career.

The Product

Earlier this year I was scanning through a forum and came across a developer who was looking for marketing help. He and a partner had written an ASP.NET invoicing package that was selling reasonably well, but they knew someone with more marketing experience could have a serious impact on sales. I looked at the online demo and I was blown away. The UI was simple, clean, filled with AJAX, and the product was easy to use.

The first thought that ran through my head? *Buy* it. An ASP.NET invoicing package was one of the ideas in my product idea notebook, and buying the application would eliminate one of the major drawbacks to leveraging a product: the initial investment to bring it to market.

So I emailed the developer to see if he'd be interested in selling.

You can hire 10 or 1,000 consultants, or sell 10,000 or 100,000 copies of your software and earn egregious amounts of money without substantially more work. But no matter how much popularity someone achieves, their hourly rate will hit a cap, and they will still work 1 hour for x dollars. This is not a bad thing, mind you, it's just that it doesn't fit my definition of leverage.

Original Location

http://www.softwarebyrob.com/2007/09/16/inside-story-small-software-acquisition-1-of-3/

The Inside Story of a Small Software Acquisition (Part 2 of 3)

When we left *Part 1*, I had emailed the developer of an *invoicing software* package, asking if he would be interested in selling the rights to his product.

Negotiations

The developer and I began an email exchange that lasted nearly a month. I received screen shots of the revenue for previous months. I reviewed code samples and data models. I played with their online demo, inquired about their customer base, how many copies they had sold, and how much time they spent supporting the product. I put it all together and, using what I know about small software product valuations, made them an offer.

After some negotiating we arrived at a final price. I received the signed contract, sent the funds via PayPal and within a few hours the code base, sales website, and domain names that make up *DotNetInvoice* (then at version 2.0) were mine.

I don't think I can properly convey the excitement I felt opening the source code for the first time; the joy of a successful acquisition. If you've ever gone through a month-long

process of investigating and negotiating a purchase you know how exhausting it is. The hours spent combing through documents and code, negotiating a price, and drawing up and signing document are all spent knowing that there is a high likelihood that the deal will fall through.

I basked in happiness for the entire evening. I called a few friends, began scribbling down ideas for new features, and gazed at the marketing website for hours. The next morning I emailed a few existing customers to inform them of the ownership change and ask for their opinion on a future direction for the product.

That's the Sound of All Hell Breaking Loose

Within 24 hours I had over 30 replies in my inbox, all of them filled with angry comments about the lack of support and the number of bugs in the software. There were issues with the recurring invoicing, unencrypted storage of data, buggy searches, and on and on.

It felt like someone had dropped a ton of wet cement on my chest…had I just spent thousands of dollars on a complete piece of crap?

Had I completely dropped the ball during due diligence?

After a few hours of panic I arrived at a game plan: email everyone, put together a complete list of bugs, and fix them.

All of them.

Every last one.

There was no doubt that this is where I had to begin to salvage the reputation of the product. At the same time I was preparing to break the news to my wife that I would be sleeping on the couch for the next 10 years until I recouped my investment.

Moving Forward

Within two weeks I had found and fixed 23 major bugs. One of the key issues was with the recurring invoicing piece. The scheduling mechanism is cleverly written to take

advantage of the caching functionality of ASP.NET so that it doesn't require a separate scheduling installation. But it didn't work.

There were a few logic errors that meant the schedules were all over the place. The bugs were hard to find and took several hours to pinpoint, but the fixes were minimal, often changing a < to a <=.

I also added detailed installation instructions (with the generous help of Joseph Voldeck from *www.MadGig.com*), since the existing instructions were geared towards experienced .NET developers. The product now includes a PDF with screen shots that walk you through the set up step by step.

With these improvements I released the next version, DotNetInvoice 2.1, as a free upgrade for all customers, even though no one had purchased support.

WHAT?!…No one purchased support?!

The previous owners offered a cheaper purchase option that didn't include support. The problem is, with or without support, when a product doesn't work you expect someone to help you. So although everyone had purchased the "un-supported" version, they (rightfully) expected someone to reply to their emails when the product had bugs.

Needless to say, within weeks of taking ownership I removed the un-supported version from the purchase options.

In addition to the weeks of development, I spent untold hours emailing customers asking them for feature requests, offering free support, and trying to regain some confidence in the product. After releasing version 2.1 I finally felt like the product was my own.

I couldn't have been happier. The new release fixed every known issue, the existing customers felt supported, and a few new sales had rolled in. Things were looking up.

Until the end of the first month rolled around.

(To be continued…)

The Inside Story of a Small Software Acquisition (Part 3 of 3)

When we left *Part 2* I had fixed 23 bugs and released the next version, DotNetInvoice 2.1, to the existing customers. The release included a fix for every known issue. Customers felt supported, and a few new sales rolled in. Things were looking up until the end of the first month rolled around.

What Sales?

The funny thing (not funny "haha," but funny like you feel after eating Sushi from a street cart in Mexico) was that after four weeks of sweating bullets, fixing bugs, and dealing with angry customers, sales were quite a bit below previous months.

Given the extensive conversations I had with the former owners I was confident their numbers were correct. But I could not for the life of me explain the sudden drop in sales.

After emailing a few customers I began to put it together: when the developers had released version 2.0 a few months earlier they had a large block of the 1.0 customers lined up to purchase it. In fact, many of them pre-purchased the product at a discount.

When the product released the flood gates opened, revenue shot up, and gobs of version 2.0 went out the door. The month I took over was one month after sales had stalled and the bug backlash began. My timing was impeccable.

The end result? I overpaid for the product. It wasn't a huge amount of money, but it did sting.

Present Day

Flash forward seven months.

With two more releases under my belt and a heck of a lot of marketing, monthly revenue is nearly triple its previous levels, and *DotNetInvoice 2.3*is a solid piece of software.

Written in ASP.NET 2.0 and SQL Server 2005, it includes source code and a 30-day money back guarantee. In the last seven months my team has implemented dozens of new features and performed some major refactors (important when selling an open source product).

We even received a *4-star review from asp.netPRO magazine*.

My Advice When Acquiring a Software Package

- **Spend a lot of time working with the code.** I'm a five year veteran of .NET and I had a pretty in-depth look at the code before making an offer. The code was clean, consistent, and worked as expected. What I didn't do was perform a complete install and thoroughly test the software (or hire someone to thoroughly test it). It would have taken several hours to get into the meat of the app and really get my hands dirty, but I trusted that since the code was clean that it functioned well. Bad assumption.
- **Email current customers.** You're spending a lot of money on a product that, even if you've spent time testing, could still have fatal flaws only noticed by someone who's used the product for months. Ask for 5 email addresses and contact customers, asking them everything: how they purchased the software, how much they paid for it, how helpful the support has been, how many bugs they've encountered, and how many have been fixed. The answers will be invaluable when extending an offer.
- **Attack the demo.** Don't be smitten by the emotional aspect of a demo (such as the glitz of AJAX or a slick design). Get in there and break the thing. Enter thirty digit dollar amounts, try a SQL injection attack, or lob rotten tomatoes at it from 10 paces. If the code was well written, the back end should gracefully handle any garbage you throw at it. If not, drop that offer by a few percentage points.

- **Investigate revenue and expenses for the previous 6 months.** Look back as many months as you can. If a month is not available assume it was terrible and adjust your offer accordingly. Take the average monthly revenue from the previous 6-12 months to eliminate seasonal or "new version" peaks. If the product is relatively new, assume revenue will drop after the first 2-3 months. If there's a lack of information, assume the worst. As a general guide, products like this tend to sell for between 12 and 30 months of revenue.
- **Fix bugs quickly to earn customer trust.** If you discover your new product has bugs, fix them quickly to show customers you mean business. One advantage to acquiring the product was that everyone was patient with me since I was learning the product along with them. Customers were surprisingly respectful because I was willing to spend the time to help them. Remember that these customers are your lifeblood. Even though they've already paid for the product, they can help you in more ways than you realize. Early on, they know the product better than you do.
- **Clean up around the edges.** It's very possible you will inherit a solid product that's rough around the edges, such as one lacking documentation or a decent user interface. Providing installation instructions or improving the look of the application are ways to impress present and future customers.

A Micropreneur's Perspective: Selling Physical Products vs. Digital Products

A recent discussion in the *Micropreneur Academy* surrounded the topic of starting an online business selling physical products as opposed to software. Since I've worked on both sides of the fence I have a lot to say on the subject.

I'm asked pretty regularly about the best way to start a physical product e-commerce site, or whether that's a good way to get started selling online.

One major benefit of selling a physical product is you don't have to build anything; the work is all in finding a supplier and putting up a site. This is great because you avoid the 200-400 hours to build something.

But there are two downfalls of selling physical products compared to software:

1. Unless you manufacture something yourself **you are a commodity**, which means price is a major issue. No matter where you go, any supplier you find will already be in use by someone else. If they aren't, within 6 months of your success you'll likely see someone else selling the same product on eBay for less.

2. As a result, **margins are tight**. If you want to stock products yourself (I am vehemently against this), you will need a garage or other large space, thousands of dollars in up-front inventory, and you will spend hours packing boxes and running to the post office…this is not work for a Micropreneur. The better route is to find a drop-shipper who will ship directly to your customers once you've made the sale. Which is great, except at that point your margins are slim…think 20-40% gross profit.

As a developer you have the ability to build/hire out/buy applications and websites, where the margin is huge and you have something unique. Whether you're selling *invoicing software* or *performance management*software your gross margin is going to be close to 100%.

But with sunglasses, paper lanterns and beach towels the margin is much smaller.

Case Study – JustBeachTowels.com

Using JustBeachTowels.com as as an example, I purchased the site because of the #6 Google ranking for the term "beach towels" and because it was priced way below its potential. I added a shopping cart and found a beach towel dropshipper (surprisingly hard to find), and sales were around $100-200 per month. Gross margin was about 30%, which meant a whopping $30-$60 per month in my pocket.

So I spent time honing PPC, SEO, a new design, a new cart, finding more dropshippers… and I finally hit on the big *low price guarantee* revelation I wrote about in my blog. Sales

shot up starting the next day and that month sales were in the low four-figures. I was pleased to say the least.

Except that four-figures in sales resulted in just a few hundred dollars in net profit after cost of goods sold.

Not bad…except I took in nearly 100 hundred orders to attain that number and I either had to place all of those orders myself or have a Virtual Assistant (VA) handle the order processing. The dropshippers are not very high-tech and don't have a way to integrate programatically with their ordering system so every order has to be placed manually.

And even at $6/hour for the VA, the cost to place 100 orders manually through a shopping cart was close to $200. No big deal, I still made a few hundred bucks, right?

Except the shopping cart (shopify.com – awesome hosted cart) charged a percentage of sales that wound up totaling about $80. And credit card processing fees took another 3%, which left me with a net profit of around 10% of my gross revenue.

From a blockbuster month of sales I made just over 10% net profit.

Why Digital Products Are Superior

This was a big lesson for me. I had spent a lot of time coddling and growing this website and all I got was a lousy 10%? This was compared to my digital product sites like DotNetInvoice, where an increase in sales results in darn near 100% of the money hitting my bottom line as profit.

It was that moment I decided to focus my effort on digital products exclusively given the huge difference in profitability.

I sold justbeachtowels.com in January of 2009 and made a great return on my investment with the goal of investing the funds into my next digital product site. Though I haven't acquired anything yet (I have my eye on a few), the return on those funds will be much higher with a software product or SaaS website.

This is not to say that you should never go into physical products; I'm sure there are exceptions to my experience, and if you stumble on a crazy deal as I did you would be foolish not to acquire something below market price. But to build a physical product e-commerce website from scratch would require much more time than it's worth given that we can build and sell software at higher margins and with less competition.

And I didn't even mention returns and lost shipments…

Original Location

http://www.softwarebyrob.com/2009/05/04/selling-physical-products-vs-digital-products/

Extras

The Five Minute Guide to Becoming a Freelance Software Developer

I started my *.NET consulting firm* with a $35 check for a business license and a drive to city hall. I didn't worry about anything but writing code and meeting deadlines. I can't say it was a bad way to go.

Until you have someone willing to pay money for your services, the tasks below are a waste of time. My advice is to be prepared to get the process started, but don't rush out and spend hours researching and implementing these steps until you're sure you have a viable business. And by a viable business I mean *you have sales*.

This advice is intended for someone looking to become a freelance software developer or web designer (or looking to start a small web design/development/consulting firm). If you intend to seek venture capital then move along…these aren't the droids you're looking for.

Disclaimer: *I'm neither a lawyer nor an accountant. I have opinions on starting a business only because I've been through it once or twice. Please seek the help of a trained professional for help with starting your business. Also, the advice about business structure and DBAs applies only in the U.S.*

Business Structure

You need to decide between an S-Corporation, an LLC, and a Sole Proprietorship.

Don't even think about a C-Corporation. They come with a ton of complexity and double taxation (where earnings are taxed twice because the corporation pays income tax on its profits, and you pay income tax on what you draw from the corporation). It's not worth the complexity unless you plan to pursue venture capital. If you do, stop reading now and head over to *Guy Kawasaki's blog*.

For the sake of the remaining 4 minutes and 42 seconds I'm not going to describe the other three options in detail when you can read about them on Wikipedia: *S-Corp*, *LLC*, *Sole Proprietorship*.

The digest version:

- A Sole Proprietorship is by far the simplest, but provides the least amount of liability protection.
- An S-Corp provides slightly more protection but quite a bit of headache at set up and tax time.
- An LLC provides the most protection and is fewer headaches than an S-Corp, but you can't grant stock options and VCs won't fund an LLC.

I run a very small shop and I'm able to track my finances using online banking and Excel. To eliminate the need for additional bookkeeping I've remained a Sole Proprietorship for five years, but I am making the move to an LLC in 2008 to reduce my liability.

No matter the direction you choose, hit *LegalZoom* to file your papers. And be sure to find a good accountant (see below).

DBA (Fictitious Name)

DBA stands for "Doing Business As," and is also referred to as a Fictitious Name because…well…it's a name you made up.

If you use your "real" name as your company name (i.e. Rob Walling), or your "real" name plus a description of your line of work (i.e. Walling Consulting), then you do not need to file a Fictitious Name statement.

If you choose to use a fictitious name for your company do an online search to ensure it's not already taken *within your industry*. Any common name you choose will likely be used by someone else, but as long as they are not operating in the geographic area you plan to target, or in your industry, you're in the clear. I would hit Google for this kind of search.

The next step is to make sure the name is not trademarked. *LegalZoom*can also help with this, or it appears you can do it on the *US Patent Office* website (although I wasn't able to figure out how to execute a search).

Next is your fictitious name statement. The idea is to place an ad in a newspaper or other periodical announcing your use of the name, and a few weeks later you'll receive a certificate indicating you went through due process. It doesn't mean you've trademarked the name; it just means you can go into a bank and open an account in the name of the business.

For this one I would also hit *LegalZoom*.

Resale License

I obtained a resale license many years ago for a comic book business I started with my brother. These days, as a software consultant, I don't have one.

If you need a resale license to buy items at a discount (some wholesalers require one to prove you're not that pesky "general public") or to avoid paying sales tax up front (since you will charge the tax directly to your customer and remit it to your local governing body), hit *LegalZoom* or call your local city hall.

Advisors

You'll need a few people in your court as you start and grow your business. You would not believe how helpful it is to have a long-term relationship with a good team of

advisors. Before I found my CPA, Lawyer, and Insurance Agent I had no idea what I was missing.

Required: CPA.

Under no circumstance are you to do your own taxes. Don't go anywhere near a copy of TurboTax. Do not skimp on a CPA. Try to get recommendations: talk to your mortgage broker, banker, or friends who own businesses. Don't settle for someone you feel so-so about. Be picky.

Depending on your locale, taxes for a Sole Proprietorship will run $400-$600. From what I hear taxes for an LLC will run $600-$1,000.

Optional: Lawyer.

I used a lawyer to create a personal estate plan, and since then I've used him as a resource. He either answers my questions or refers me to someone who can.

Optional: Insurance Agent.

I have an insurance agent I use for personal insurance needs, who has come in handy on many occasions when discussing my business needs (liability insurance, etc…).

Health Insurance

If you have a working spouse by all means take advantage of their insurance. If not, see if *Kaiser* is available in your area, or hite*HealthInsurance*. Health care is absurdly expensive, so be prepared to pay $500-$1,000 per month for decent family coverage.

Other Insurance

It's common to wonder if you need Errors and Omissions insurance. This depends on the kind of clients you'll be working for and the likelihood that you will be sued. Most solo developers I know do not have E&O insurance, as it runs $1500-$2500 per year. It's up to you to decide how comfortable you are with or without it.

Another option is to get a personal umbrella insurance policy that provides a pre-specified amount of coverage if you are sued. This type of coverage will only help you if you are a sole proprietorship.

Banking

This is critical: open a business checking account using your company name. This allows you to write checks with your business on them (does anyone write checks anymore?), to cash checks made out to your business, and to easily set up a PayPal or merchant account.

Do not skip this step.

Not only is it a good check to ensure you've set up your business correctly since a bank will not allow you to open an account without the proper business documentation, but it's one of the first questions the IRS will ask when you try to write-off business expenses.

Credit Card

Get a business credit card to track expenses. *Advanta* has screaming low rates, great cash-back percentages, and they're geared towards small businesses. You can even get additional cards with spending limits, and all purchases are tracked separately for each card.

Get an extra card for your spouse and put a $100 spending limit on it. He/She will love it.

How Much to Charge

Sigh. So much has been written on this subject. The right answer to this question is: what the market will bear. Do your research. Until you've differentiated yourself you will be at market rate, which is probably less than you're worth if you're taking the time to read this post.

As you invest in *self-marketing* this rate will rise as people come to you instead of the other way around. The first time this happens you will fall out of your chair. Take it as a sign of demand.

One mistake I see many freelancers making is charging too little. If you always have too much work then raise your rate until you have a more reasonable workload.

Retirement

I'm not going to guilt you into this one, but if you are considering becoming a freelancer and not putting away at least 10% of your annual income into a retirement account, email me and I will convince you why this is a terrible idea. Bottom line: *save for retirement*.

Vanguard is leaps and bounds above other investment firms I've used – they have low rates, excellent performance and amazing customer service. I honestly don't know how they do it.

Read up on the *SEP IRA* and *SIMPLE IRA*. I've used both and they are both great tax shelters.

Source Control

You don't want to be a server administrator. With the number of patches coming out these days, initial configuration, backups, upgrades, repaves…using an old desktop as a source control server is more trouble than it's worth. It will run twice the cost of a hosted solution in maintenance hours alone.

And don't forget what happens when something breaks and you don't have time to fix it. *This will happen*.

My recommendation for those seeking *simple, cheap source control* is to open an account with *DreamHost* (careful, I get some referral money if you click on that link) and use their free Subversion add-on. I've been using it for going on 18 months and it's *brilliant*. Fast, reliable, no hassles.

If you have to use a Windows-based solution such as Vault or VSS, get a Virtual Private Server (VPS) from someone like *HostMySite.com*. A Windows VPS starts at $40/month and I've had nothing but good luck with HostMySite.

Time Tracking

I have two favorites: *SlimTimer* and *ClickTime*.

Bug Tracking

I've tried several free tools and couldn't get past the learning curve or the maintenance issues. I have finally settled on Hosted FogBugz and have never once regretted the decision. Talk about great customer service and painless bug tracking, collaborative spec authoring, customer support (for my Micro-ISV), project management, software estimating, etc… It's not cheap, but the time and aggravation you save will pay your FogBugz bill 10x over.

My five minutes are up! If you have questions or comments you know what to do.

Original Location

http://www.softwarebyrob.com/2008/01/18/the-five-minute-guide-to-becoming-a-freelance-software-developer/

Epilogue

If you enjoyed this eBook, please let me know on Twitter (this helps me know if I should do things like this again in the future):

@robwalling Enjoying your free 'Best of Software by Rob' eBook: *http://bit.ly/hY0p6y*

To ask a question, make a comment or download a copy of this collection, visit *www.SoftwareByRob.com*.

Until next time.