
START MARKETING THE DAY YOU START CODING
HOW TO MARKET YOUR STARTUP, GROW YOUR COMPANY, AND THINK LIKE AN ENTREPRENEUR.
ROB WALLING

START MARKETING THE DAY YOU START CODING

HOW TO MARKET YOUR STARTUP, GROW YOUR COMPANY, AND THINK LIKE AN ENTREPRENEUR.

ROB WALLING

Cover Design & Interior Design by Ari Constantino.

RobWalling.com

TABLE OF CONTENTS

INTRODUCTION

I published the first essay on my blog in June of 2005 with a handful of bad posts and the uninteresting mission of "explaining 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.

Today, the term "blog" sounds dated similar to Web 2.0 and DHTML. But in the 00s, pre-social media, blogs were how we communicated what we were learning. This later spread to podcasts and Twitter, but blogs had a really solid run for a few years.

During the 6 or so years mine was active, I published 383 posts, received over 2,000 comments (+10x that in spam), and received well over a million visits from 211 countries.

While looking through my archives back in 2011, I realized that many posts were as relevant as they had been on the day they were written (whether that was three days, three months, or three years prior). So I decided to collect them into a portable, well-formatted, and easy-to-read version containing only the best essays.

Then in 2023, I resolved to breathe new life into this book by releasing this revised version, updating much of the text and adding newer essays that were published after the first edition of this book. I hope these articles will have the same impact today as they did on the day they were published.

Rob Walling
May 25, 2023
RobWalling.com

P.S. If you enjoy this book and want to hear how I've been thinking about startups since 2011 you can find me in two places. My podcast, Startups for the Rest of Us, and Twitter, @robwalling.

01

STARTUP MARKETING

LOSING PEOPLE THROUGH THE BOTTOM OF YOUR FUNNEL [1]

I had a realization recently while talking with a MicroConf attendee: 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*

---

1 Original location: https://robwalling.com/2009/12/15/startup-marketing-mistake-losing-people-through-the-bottom-of-your-funnel/

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 your 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 your conversion rate was high, and you will never have an easier time driving traffic to an application than 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 the time and effort you spent sending them to your website and converting them into customers.

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 a higher priority than improving your conversion rate, and definitely a 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 anyone 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.

WHY FOCUSING ON TRAFFIC CAN KILL YOUR STARTUP[2]

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 websites. Almost no one thinks about improving conversion rates. *Why is this?*

Good Marketing Looks Easy

Good marketing, as with good design, looks easy. So easy it's virtually invisible.

---

2 Original location: https://robwalling.com/2009/12/09/why-focusing-on-traffic-can-kill-your-startup/

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 subject 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. 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:

1. Build some traffic. A few hundred visitors per month, maybe a thousand

2. Improve the conversion rate

3. Send more traffic

4. Improve the conversion rate some more

5. Send more traffic

6. 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.

THE NINE LEVELS OF TRAFFIC QUALITY[3]

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

---

3 Original location: https://robwalling.com/2010/01/12/startup-marketing-mistake-ignoring-traffic-quality

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% difference in MicroConf sign-up conversions based on the source (and thereby the quality) of the traffic.

The reason this is important is that 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 MicroConf attendee. 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 for 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:

1. Mailing list (assuming you have a relationship)

2. Your blog (assuming you have a relationship)

3. A referral link from a targeted website with a positive write-up about your product

4. 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)

5. An organic search on your product name

6. A referral link with no write-up or from a non-targeted website (such as TechCrunch)

7. All other organic searches (assuming this is their first visit)

8. Google AdWords

9. 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.

THE #1 GOAL OF YOUR WEBSITE [4]

This post is an adaptation of a portion of my 2010 Business of Software *talk.*

What's the #1 goal of your website? Ask this of 10 software entrepreneurs and you'll hear the same answer 10 times: the #1 goal is to sell software.

It's a nice guess…but it's wrong.

The #1 goal of your website is not to sell software. Not unless your purchase price is less than the "impulse" price point of around $20. If it's above this mark then most people will not buy on their first visit to your website.

And if you pressure them with heavy-handed marketing, you'll give them an inexplicable feeling of pressure and discomfort. Asking someone to buy before they have confidence in you or your product is a recipe for abysmal conversion rates.

The ineffective marketer asks you to buy too soon.

---

4 Original location: https://robwalling.com/2011/01/12/the-number-one-goal-of-your-website/

And by “ineffective” I mean that no one buys from them. This is the same guy who asks you to buy life insurance the first time you shake their hand at a party, or posts affiliate link after affiliate link to their Twitter stream. Unfollow.

Very few people will spend more than around $20 the first time they visit a website. As you move into (and beyond) the $30 range your conversion rates drop dramatically on first-time visitors making a purchase. In this case, you need them to come back to your website multiple times in order to close the sale.

Why People Don’t Buy
There are five basic reasons people don’t buy:

1. They need more information (does it have the features I need?)

2. They don’t trust the product or company

3. They don’t have the budget

4. They don’t need it yet

5. Never going to buy (just researching)

We can’t change #5, but the first four can be changed over time through repeated contact. In fact, looking at how many obstacles people have to overcome in order to make a purchase, it’s amazing anyone buys on their first visit to your site, at any price point.
I’m a Numbers Guy

I have an expression I like to use:

Data is my co-founder

Since I’m a solo founder I can’t rely on a conversation with my business partner to help make things clear. I have to rely on data

to guide most of my decisions.

So let's take a look at some numbers to try to back up this assumption that people don't buy on their first visit to your site. There's a common misconception that your conversion rate is a single number. The single number people talk about – that magical 1% number – is actually an average of all of your individual conversion rates.

So let's break that average up for a few actual products:

- Over a three-year period, DotNetInvoice made 450% more sales to returning visitors than first-time visitors
- Over a one-year period, JustBeachTowels (a website I used to own) made 770% more sales to returning visitors
- Over about a three-month period, AWPCP (an Academy member's product) made 400% more sales to returning visitors
- Over a 60-day period, CrazyEgg made 1585% more sales to returning visitors

As an aside: you should absolutely be tracking these numbers on your own site. It's as simple as setting a purchase goal in Google Analytics. It takes 30 seconds and will provide you with some of the most actionable data you can glean from an analytics package.

Is This Proof That Returning Visitors Are More Valuable?
Of course not. Proof would require reams of data and more processing power than a Knights Ferry.

But it *does* provide a good rule of thumb that you should use to examine your own sales patterns. In my experience selling this holds true in *almost* every case.

The other shocking fact, when coupled with the above, is that the

vast majority of your website visitors are first-time visitors. Typically 80% or more. Egad! Do you realize what this means?

It means that your largest pool of prospects is hitting your website and taking off without returning. But the portion of them who return is much more likely to buy your product. According to the numbers above, they are 4-16x more likely to buy.

Ok, Then What's the Real Goal of My Website?
Your goal should be to get first-time visitors to come back.

And not just come back, but have repeated, "confidence building" contact with your company or product. It's not your job to shove your $300 product down their throat on their first visit, but it is your job to convince them that coming back is worthwhile and to help them overcome the four objections I listed above during the process.

How can you do this?

You could use RSS, Facebook, YouTube, LinkedIn, Twitter, or email. All are good options, but each has its flaws.

RSS for example, is used by a very small percentage of Internet users. Extremely small. Ask your mom what RSS is and she'll tell you it's an acronym for a hallucinogen she tried in the 60s.

Facebook, YouTube, LinkedIn, and Twitter are all new, hip, and cool…but a 5,000-person following in any of these venues is worth much less than a 5,000-person email list. Email has so many things on its side:

- Ubiquity
- Less time commitment needed to maintain it
- Highest click-through rate of all options

- You can control what each individual user sees on any given day

For all its foibles, and all the techie hatred of email, it's still head and shoulders above any other way to connect with most of the online world. My mom barely has a Facebook account, never goes to YouTube, has never hear of LinkedIn, and only knows Twitter from TV shows…but she's been on email every other day for 10 years.

Don't convince yourself that just because you don't like email the rest of the world feels the same way. It's not the case.

Social media routes (including blogging) are all fantastic avenues for building an audience. And if you have the time, tools, and talent to make it happen then by all means get to it! But if you want the fastest, optimal way to connect with most audiences, it's through an email list. Email should be your first goal, with social media approaches coming once you have this cornerstone cemented in place.

So how can you get started gently engaging customers through email in a way that won't commit you to spend hours every week managing and creating new content for it?

Step 1: A Killer Landing Page
People don't like to give out their email. You have to first build enough trust that they are willing to provide it. The easiest ways to build confidence are through solid web design that puts the email sign-up at the forefront of the page, or by providing information that shows you are an expert.

Examples of solid web design that give you the confidence you're not going to get spammed are here and here. Notice the prominence of the email sign-up forms.

For a visual look through tons of landing pages (both good and

bad), try this Google image search.

Step 2: Give Something Away
You can give away an 8-12 page PDF report or whitepaper. This works, especially with a great title "6 Things You Must Know Before Implementing a CRM System" or "8 Expert Tips for Working with PDF Documents." These will convert and they're a good start.

But the key to high conversion rates is doing something unique. Jump Start's PDF is awesome. I didn't even need the information it covered, but the design is so freaking cool I downloaded it anyway. This spread virally and sent a lot of traffic to this landing page.

Smart Bear gives away an ebook written by Jason Cohen. They send out 1000 a month.

Step 3: Automate Your Follow-Up
Finally, set up an automated follow-up sequence. This is a short series of emails that are sent to the subscriber over the course of a few weeks or months that provides information, perhaps a discount, and builds trust in your product.

After using four email service providers over the years I am a firm fan of MailChimp due to their superior feature set, reliability, and customer service. They also have a free trial that allows up to 1000 subscribers so you have a long runway before you have to pay them a dime.

I'm not going to go into the specifics of setting up a subscriber form on your website since it's essentially copy-paste of their HTML into your web page, but once that's in place you'll want to set up an auto-responder that provides valuable information a new subscriber at regular intervals.

What kind of intervals, you ask? Let's take a look at two follow-up

sequences that have worked for real software products.

Collectorz.com ran a split test on requiring an email before downloading a free trial, and not requiring the email. The process is documented here, but the summary is they sent a 4-step follow-up sequence to anyone who provided their email:

- **Day 0:** Welcome, $5 discount coupon, download link, link to Getting Started guide, testimonials.
- **Day 2:** Buying guide, standard vs pro, barcode scanners, iPhone app, testimonials.
- **Day 6:** Subscribe to newsletter invitation, benefits of cataloging your stuff, testimonials.
- **Day 30:** Outright invitation to buy, more testimonials.

And collecting emails resulted in the following:

- **#Downloads:** down by 33.6%
- **#Sign Ups:** up by 20.7%
- **#Sales:** up by 3.4%
- **Average First Purchase Value:** up by 13.5%
- **Profits:** up by 15.4%

In a second example, software vendor SolarAccounts increased sales through the same process.

To summarize, they sent two emails over 14 days:

- **Day 0:** As soon as the user downloads the software, we send an email saying, “Thanks for downloading – here is a useful

How-To guide"

- **Day 14:** Two weeks later we send another email (if they have not opted out) asking "Did you like our software?"

The results?

- 15% fewer downloads
- **18% increase in sales**

Do these two split tests unequivocally prove the point that asking for an email and reminding prospects results in higher revenue? Nope.

But it has to make you curious to the tune of the couple hours it takes to run your own split test using Optimizely, Visual Website Optimizer, or Google Website Optimizer.

It'll take a few hours of work over the course of a few weeks, but the odds are good you can increase sales by 10% or more based on the examples above, and experience with my own products.

WHY "LUCK" IS A TERRIBLE MARKETING PLAN FOR YOUR STARTUP.[5]

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.

---

5 Original location: https://robwalling.com/2009/09/22/why-luck-is-a-terrible-marketing-plan-for-your-startup/

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 throw @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 provides 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 of 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 chance 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: Entrepreneurship 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 involves 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 months" 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.

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.

WHY FREE PLANS DON'T WORK [6]

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, and 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)."

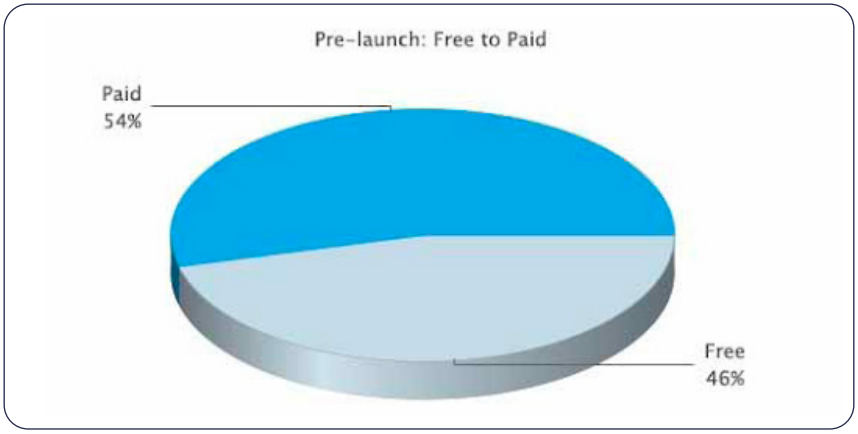
---

6 Original location: https://robwalling.com/2010/08/18/why-free-plans-dont-work/

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 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. This 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 which 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 then I 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 up front. 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 regard 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 for 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 was a Freemium Twitter web app created by Justin Vincent. It had a great stats page that showed everything from monthly revenue to the breakdown of users by plan type. Taking a look at that page revealed that he was actually doing very well for a relatively young app in this space.

He had been averaging about a thousand dollars a month. And unlike the bigger guys, his paid users made up 2.5% of all accounts. That was damn good for any sort of Freemium app judging by the numbers that we had 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 had been relatively flat, and the number of users had 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 on 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 entrepreneur.

WHY MAKING SOMETHING CUSTOMERS WANT ISN'T ENOUGH [7]

In his essay *How to Start a Startup*, Paul Graham famously stated there are three things needed to create a successful startup:

- To start with good people
- To make something customers actually want
- To spend as little money as possible

This list is pure genius. However, over the past several years running my own businesses and working with hundreds of entrepreneurs, I've found a fourth ingredient that's necessary for a company to succeed:

Customers must give you more money than what you spend to acquire them.

I realize this is an obvious statement. But here's the kicker…

Once you understand that a customer will generate a specif-

---

7 Original location: https://robwalling.com/2011/07/21/why-making-something-customers-want-isnt-enough/

ic amount of revenue during their entire relationship with you (called your lifetime value of a customer) and that you want to spend less than that amount to acquire them, this leads to a more practical point:

How do you know, in advance, how much it will cost to acquire a customer unless you know how you're going to do it?

Let's think about this for a moment. When most of us think about a startup we think about the sweet new technology we're going to use, we (hopefully) talk to customers to find out what they really need, or we size our market using a top-down or bottom-up approach depending on how realistic we want to be.

(Hint: if you're looking for funding, use top-down. If you're bootstrapping, use bottom-up.)

But we rarely think about the specifics of *how* we are going to reach prospects, and how much that's going to cost. The reason is that the answer is pretty scary in most cases. And nearly impossible to measure in others. It's a lot of guesswork until you really dig in and actually try the marketing approach yourself.

A Look at One Acquisition Approach – Google AdWords

Since this will be easier to explain with an example, let's start by looking at the cost of acquiring a customer using Google AdWords.

If you're bidding on a term like "invoicing software" you might have to pay around $4 to rank in the top 3. If you can convert 1% of your visitors to customers this means you need 100 clicks for each purchase, making your cost per acquisition (CPA) $400.

If, however, you convert at 0.5% your CPA jumps to $800. Ouch… this is why you can't make money charging $1 per month.

Your lifetime value of a customer (LTV), cost per click, and conversion rate are critical in figuring out if you can build a profitable business or if the competition is too strong. The problem, of course, is you can only obtain one of the three numbers needed before you begin marketing your product. But you can make an educated guess at the other two.

Calculating LTV

If you have a product with a one-time purchase price (something like DotNetInvoice, which sells for $329), your conservative estimate is to assume your LTV will be that purchase price minus payment processing fees. I know you'll be adding add-ons, upsells, and annual maintenance plans.

But in my experience, these account for a small bump in LTV unless they convert well, which is the subject of another post.

If you own an application with a recurring pricing model, you need to know your price point and churn rate for each of your plans in order to calculate your LTV.

Churn rates are all over the place depending on the growth stage of the company, the industry, etc… When you're just starting out you'll probably have monthly churn in the 5-10% range. Some larger SaaS firms get their churn below 1% per month.

Taking monthly churn at 8% and your monthly price at $19, your lifetime value of a customer works out to $237.50. This is calculated using the following simple LTV equation (there are more complex formulas for LTV that take expenses into account):

LTV = price/churn
LTV = $19/.08 = $237.5

With this number in mind, Google AdWords is not going to work unless you like buying high and selling low.

As an aside, even if your CPA using AdWords was $100, you would still need a pile of cash to finance your customer acquisition process since your lifetime value will take 12.5 months to arrive in your bank account. This is one case where venture funding makes complete sense.

How About a Few Others?

Facebook
Most Facebook clicks run in the $.80-$1 range with no effort, but using tricks I mentioned in this guide to cheap startup advertising you *can* get clicks in the $.10-$.20 range.

If you convert 1% of traffic with an average cost per click of $.40, your CPA will be $40.

If you convert 0.5% you'll be at $80.

SEO
SEO is a tricky one since your investment will pay out over time rather than send you a few clicks and die. If you can get in the top 3 for a keyword that will send you 200 clicks per month with minimal maintenance, but you need to spend $1000 (or a lot of up-front time), would you do it?
If you can convert that traffic at 1% you'll make 2 sales per month. If your LTV is $250 per customer this is $500 in future revenue per month. As long as you have the cash to fund it, this is quite an investment.

Cold Calling
If you decide to go the cold call route you're going to make 10 or 100 calls yourself to develop a script, but after that, you're going to hire a service to make the calls in order to keep your business scalable. After setup fees, telemarketing services charge $15-$20 per hour. If they can make 20 calls per hour and close 1 out of 100 calls, you're looking at a CPA of $75-$100 depending on the hourly rate.

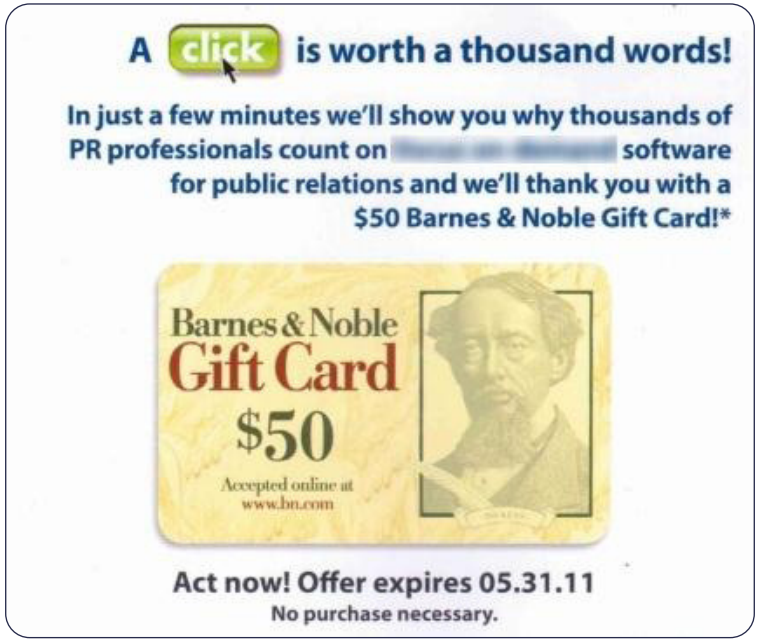
Note: I've never used a telemarketing service and the numbers above are estimates only. If you have more experience with real costs and conversion rates, please post a comment.

Word of Mouth

Word of mouth is free unless the referral requires a high-touch sale. If you can encourage word of mouth with a healthy referral bonus, you can see pretty easily how this can work in your favor.

If your LTV is $250 and someone refers a new customer, sending them $50 or $100 is an easy sell. This explains how HostGator can pay up to $125 in cash for a referral; they know their LTV.

It also explains how companies can afford to give you a $50 gift card for attending a webinar. They know their LTV and their conversion rates from the webinars. From there it's simple math to determine what they can put on that gift card and still make a profit.



Conclusion

There are hundreds of approaches to acquiring customers. The point of this post is not to explore the economics of everyone, but to get you thinking about the financial nuts and bolts of your marketing before you build your product.

Building something customers want is a critical step in the process of launching a successful startup. But if that's your only metric of success, you may be unpleasantly surprised when you find out you can't acquire a new customer for less than $250 when your LTV is only $150. Unless you're doing dotcom math you'll be out of business in no time. (BTW – If you are doing dotcom math you'll have venture funding and be a billionaire in no time).

My hope is that this post encourages you to take your best shot at estimating your LTV and CPA before launching your startup and, better yet, encourages you to get out of your comfort zone and actually try some of your proposed marketing approaches before writing a single line of code. (BTW – if you're looking for info on how to do that, I talk about it in chapter 2 of my book).

YOUR MARKET IS SMALLER THAN YOU THINK [8]

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 online and perhaps

8 Original location: https://robwalling.com/2010/09/30/your-market-is-smaller-than-you-think/

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 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 the 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.

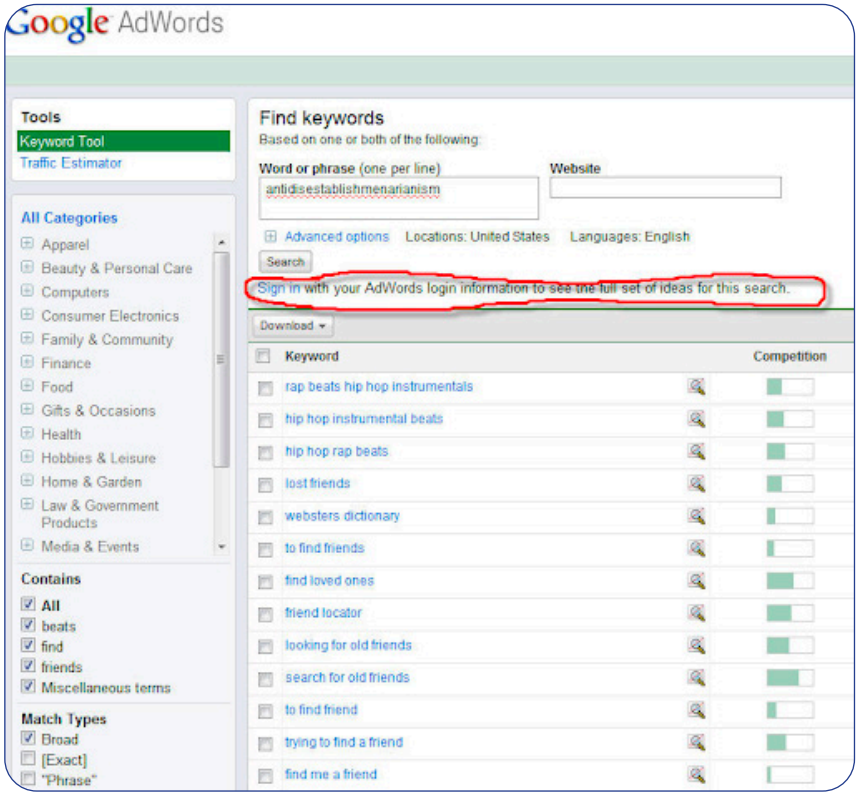
Mistake #1: Not Logging In

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 whizzbang 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.



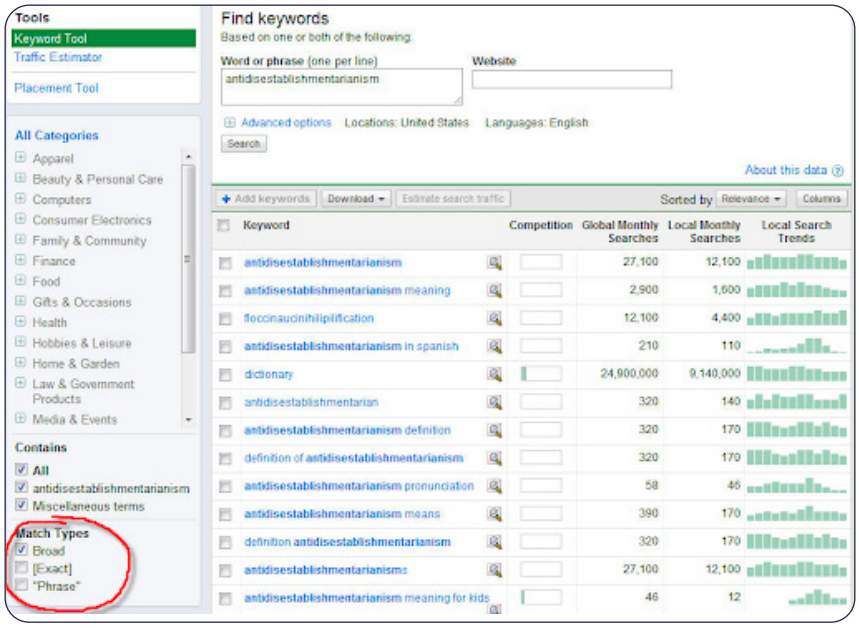
Mistake #2: Using the Default Match Type
Next, enter your term and hit "search."

"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.



You don't want a broad match…you want an 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	27,100	12,100

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.

WHY YOU SHOULD START MARKETING THE DAY YOU START CODING [9]

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 MicroConf 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.

---

9 Original location: https://robwalling.com/2010/10/14/startup-marketing-part-6-why-you-should-start-marketing-the-day-you-start-coding/

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 I 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 is 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 article.

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 set up 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 1 am. 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 to 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 addresses. 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%, 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 about theory without giving actionable advice. So let's take a quick look at the details of getting set up 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
- Set up 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 set up the major task is driving traffic, which is beyond the scope of this article (but I wrote about it in my book).

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, and 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 screenshot 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 of 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
- 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.

HOW I CREATED 4 STARTUP EXPLAINER VIDEOS FOR $11 [10]

When I was writing the copy for the home page of Drip, I ran into a bit of a challenge: the idea of marketing automation is still new to

10 Original location: https://robwalling.com/2015/02/17/how-i-created-4-startup-explainer-videos-for-11-dollars/

a lot of people, so not everyone is aware of how email marketing (much less marketing automation) can help them.

One of the easiest ways I've found to explain the value of Drip has been making screencasts, walking people through the product, and pointing out very specifically how Drip can help in their particular use case.

But making personalized screencasts for every trial customer isn't realistic, so I put my mind to finding a way to make this a little more scalable. After discussing with Derrick we came up with the idea of creating a few explainer videos to walk visitors through how I would tag, segment, and structure a Drip account for a few different business types.

Normally, I prefer to use contractors (typically those who offer productized services) for basic marketing elements so I can focus on higher-level tasks, but in this situation, there was a catch…

We have four distinct customer types that find Drip valuable, but they're attracted to the product for very different reasons. And at roughly $2,000 apiece, hiring out production of 4 videos was going to get expensive in a hurry.

So I came up with a process to create multiple explainer videos quickly, with the help of a couple of inexpensive services. In the end, my total out-of-pocket cost was $11.

But here's the catch: in addition to the $11 dollars, this process took about 8 hours of my time. If you're a funded startup or a top-dollar consultant (or you only need 1 video), you should probably just hire it out and save yourself the time.

But if you're a bootstrapper or need multiple videos as I did, it makes a lot more sense to spend a day putting these videos together than shelling out $8000 dollars to have someone else handle it.

So without further ado, here's the process I used to create 4 explainer videos in about 8 hours, for $11).

Step #1. Write the script (Estimated time: 45 minutes)
To kick things off, I wrote a script for the first video. It wound up taking a form quite similar to a sales letter, starting with a narrow focus to grab the visitor's attention, then expanding on the details of what Drip can do.

Sometimes writing a clear, persuasive script can take hours, but this first one took me about 45 minutes – partly because I'd had so much experience explaining the product in screencasts, sales calls, and on the long-form homepage, but also because explainer videos need to be super short (so there wasn't much copy to write).

Your video should not be any longer than 90 seconds, and to be honest, the shorter the better. If you can make a case for your product in 45 seconds, that'd be great, but for most people, the video will end up being at least a minute long.

Try not to worry about duration while you're writing, but keep in mind that you need to be clear, precise, and straightforward.

Here's how it turned out:

> *Meet Frank*
>
> *Frank is interested in your ebook, software or other digital product, but he's not quite ready to buy.*
>
> *Instead of leaving your website and never coming back, Frank notices you're offering a free demo version or sample download from every page of your website. Made possible using one line of JavaScript from Drip.*
>
> *So Frank gives you his email address and downloads*

your sample.

Over the next few weeks, Drip sends Frank amazing educational content, convincing him you're an expert in your space. And when he's ready to make his buying decision, it's a no-brainer for Frank to buy from you.

If you use one of these payment providers, in addition to several others, your Drip account knows Frank became a customer without you having to write a line of code. And if not, it's as simple as adding a single line of JavaScript to your website.

In Drip, you've setup an automation rule for purchases, so Frank is automatically moved from your lead nurturing email campaign into your customer campaign that follows up, provides more educational content, and ultimately leads to Frank buying more of your products.

You also tag him as a Customer so you can easily send one-off promotional emails when you're having a sale, release a new version, or just need a quick influx of cash.

Drip is unlike any email marketing platform you've ever used, and it can handle every phase of your customer's lifecycle, from website visitors to sample downloaders to paying customers.

All this and so much more is possible with just a few clicks, with Drip.

Step 2#: Walk Away for a Day, Then Edit (Estimated time: 30 minutes)

In order to give the copy a solid revision I put it in a drawer for a day. Coming back for a "cold" read highlighted a number of flaws

right away.

After the first round of revision, I started reading the script out loud, which brought out a few more flaws right away.

With those fixed, I edited for length – I think I was around 1m55s, so I cut a paragraph or two to reduce the length to under my self-imposed 90-second limit.

Step #3: Record the Audio (Estimated time: 15 minutes)
This is one of the easier parts – I used a free piece of software called Audacity to record and edit the audio (you could use Garage Band or any other recording software), as well as a high-quality mic. I used the same headset I use to record my podcast since I know it's up to par, but it's important that the mic connects through the USB port and not the little headphone jack; the audio quality is much better on USB mics.

Don't put yourself under any pressure to nail your first take here…just hit record and read through the script. Any time you mess up, start the sentence over and cut the bad sentences out during editing.

Editing is what takes most of the time here, but since you're just cutting out sections of one audio track, it's still relatively fast.

Render the file as an MP3 and you're all set.

Step #4: Turn the Audio into Visuals (Estimated time: 1-4 hours)
Turning ideas from my recording into physical images was the longest part of the process by far. I knew I wanted to find drawings that I could move around to "animate" the video, but it's hard to find high-quality drawings to license and even harder to decide how to visually represent technical terms.

For instance, when I mentioned "Frank" visiting your website and being tagged, it took me a while to track down an image of a guy

with a laptop and a question mark next to his head and to come up with the idea of using a luggage tag.

This is the creative part of the video, though, so it's worth the time investment and in retrospect, I'm glad I took the time to search as long as I did to find decent images.

Stock image sites tend to have a large collection of clip-art style drawings, and 123RF, in particular, has a lot of drawings that you can search before signing up.

I was surprised at how long this section took (for me, it was close to 4 hours). That included finding them, downloading, resizing, printing them out, and cutting them out.

Step #5: Filming (Estimated time: 1 hour)

Once I had all the drawings printed out, I grabbed some butcher paper from my kids' art closet to use as the backdrop.

If you have access to high-end video equipment, that's great, but it's not necessary – I just propped up a consumer-grade HD camcorder. I didn't even have a tripod for the first videos (which was ridiculous, in retrospect…I bought one the next day before I recorded the remaining 3 videos).

It probably took me 10 takes to get the first video right. Timing is definitely the hard part because I was listening to the audio recording while trying to move the pieces around in time with the words. It took some practice and coordination to get to a point where it didn't look ridiculous, but the final product turned out great.

Step #6: Editing & Rendering (30 minutes)
I edited my videos in iMovie – it isn't the world's best video editor, but it works (and if you're on a Mac, it's free)

Then I rendered and exported it as an MP4 and voila!

Here's the finished product.

Step #7: Repeat
I made sure to focus on getting one video right before I worked on the others. Once I was really happy with the first version, then, and only then, I went back and did the next three.

This was to ensure that if I made a mistake in one of the later steps, it only screwed up one video, not all four of them.

By practicing that way and ironing out a process, I made all my mistakes in the first video.

Once I had the process and coordination down, I cranked out the other three.

I wrote the scripts for the next three videos, used basically the same clipart, and recorded all three in under two hours.

What's next?
This process may not be the best fit for everyone, but it's an inexpensive way to build yourself a decent explainer video.

My videos have worked out really well for Drip. I've used them in countless email exchanges to quickly explain Drip in terms that are very specific to their niche. And around 25% of home page visitors watch one of them.

Would I have gotten prettier videos if i dropped $8,000 bucks on them?

Probably.

But my videos work for now, so it makes more sense to put that budget toward other marketing approaches (or building new features).

And, of course, I'd love to see your videos if you follow this process. Just ping me on Twitter: @robwalling.

HOW TO FIND YOUR 4-SECOND STARTUP PITCH [11]

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

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 grabs the reader 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

11 Original location: https://robwalling.com/2010/05/13/how-to-find-your-4-second-startup-pitch/

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:

1. Explain what your product does and for whom. Such as "Simple proposal software made for designers."

2. Make a promise to the customer espousing a benefit of your product, such as "Save Time. Get Paid Faster."

3. 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:

1. **What it does and for whom:** "Inventory Tracking for Grocery Stores"

2. **Promise:** "Automate Your Inventory"

3. **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 simple; 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 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.

HOW TO COMPETE AGAINST OPEN-SOURCE COMPETITION [12]

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

12 Original location: https://robwalling.com/2009/08/11/how-to-compete-against-open-source-competition/

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 recommended it has run into major compatibility issues or a 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 their 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; the odds are high that 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.

EXPENSES YOU DON'T THINK OF WHEN STARTING A STARTUP[13]

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,

13 Original location https://robwalling.com/2009/01/07/expenses-you-dont-think-of-when-starting-a-startup/

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 start-up 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 to 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 it 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.

- **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 website.

- **Phone** - $20/month for a landline. $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 backyard and shot, but I still send a couple of faxes each month. eFax will run you $14/month (annual plan) or if you have a landline 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. 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.

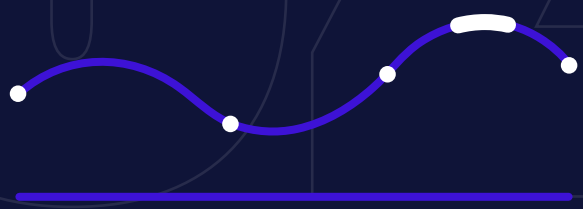
- **Software** - If you're one of those lucky Ruby or PHP developers then most of your tools are free.

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.

02

THE ENTREPRENEURIAL MINDSET



FIVE REASONS YOU HAVEN'T LAUNCHED[14]

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.

---

14 Original location: https://robwalling.com/2010/11/10/five-reasons-you-havent-launched/

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 skill set, 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 rehash 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 works 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).

HOW TO FORCE YOURSELF TO SHIP (EVEN WHEN IT'S HARD)[15]

It's been a week and a half since we launched Drip's biggest feature in 18 months, called Workflows.

Had we not committed to a deadline, in public, 2-weeks prior to the ship date (when we published a post on Drip and emailed a bazillion people), I believe we'd still be adding finishing touches.

It's scary to ship. And it's in our nature to want everything to be perfect before doing so.

When working in tandem, these two forces can keep you from shipping for far too long.

Enter Deadlines

This is why I've gone back and forth over the years about deadlines. Deadlines, especially those you commit to in public, are great at *forcing* you to ship.

But they can also have a negative impact on your quality of work (if you don't give yourself enough time to do things right), or your

15 Original location: https://robwalling.com/2016/02/09/how-to-force-yourself-to-ship-even-though-its-hard/

quality of life (if you don't give yourself enough time to do things right).

So personally, I don't use deadlines all that much. But every once in a while (and especially when you're just starting out), the discipline of a deadline works better than perhaps any other motivational tactic to force your hand and make you ship.

And with our Workflows launch, the deadline worked.

It forced us to make some tough decisions and get things out on time. And the launch turned out to be our biggest growth accelerator of the past 18 months. Hooray for shipping.

But committing to a public deadline is pretty scary. Here's why you should do it anyway…

What You'll Learn

If you're doing things right you should be shipping a *lot* more than feels comfortable.

Especially in the early days, you should ship something monthly. Weekly. Maybe even multiple times per week (think of starting a blog and posting 2 or 3 times per week).

If you're not doing this early in your journey, you're not going to build up your *shipping muscle.* This is the muscle that's weak and frail in all of us when we begin. And you strengthen over time through shipping, and shipping alone.

Shipping is also the event that gives you the *rush*.

This is the endorphin infusion you experience when hitting "deploy" on a new feature. Hitting publish on a blog post. Or launching an ebook on which you've slaved away for months.

The rush is the thing that keeps you going, yearning for the next

time you can experience it. And if you wait too long, you forget how good it feels.

Fear
Don't get me wrong, shipping is scary. It gets easier over time, but the fear never goes away completely.

I believe that our desire to not ship is created by multiple fears working in tandem with one another:

- Fearing people will criticize you
- Wondering if what you've shipped is good enough
- Wondering if anyone will notice
- …and secretly hoping no one does

These are the big ones you'll feel every. single. time. you. ship.

The answer? Strengthen your muscle. Ship more often.

It makes the terror decrease (just a bit). And it fires those endorphins that make you remember why it is you decided to embark on this journey instead of keeping your day job like a sane person.

How to Keep Shipping
Back to deadlines. A great way to force yourself to ship is to make an ongoing public commitment. Start a weekly podcast, for example. It'll put your feet to the fire.

I'm pretty sure this is why Andrew Warner has shipped 5 interviews a week for years.

John Lee Dumas has done 7 per week.

Even schlubs like Mike and I have pushed out an episode every

week since 2010.

I can't speak for Andrew and JLD, but Mike and I are not particularly well organized. We don't have our act together much more than the next person.

But after pushing a few episodes you start to feel like you've made a commitment to keep doing what you've been doing. It starts with a few weeks, then it's a few months, then a few years.

By the time you get there you don't want to break the streak. You're a little scared to see what would happen if you missed a week.

Shipping is not only addictive, but it builds momentum and makes it easier to ship next week. And the next. And the one after that.

Public Deadlines

Maybe you don't want to make an ongoing commitment, and just need motivation to get a big feature (or product) out the door. Your other option is to go old school and set a deadline. Nothing new here; we've all had them before.

But when was the last time you set a deadline for yourself and *didn't move it* when things got tough?

I'm not talking about a deadline set by your boss, spouse, or local tax authority. I mean sitting down, choosing a realistic date. And, come hell or high-water, shipping by that date.

Even if your kid gets sick. Even if the next episode of *Game of Thrones* comes out. Even if you realize you missed a major piece of functionality and are going to have to scramble to fit it in.

Doing this will be both stressful…and exhilarating.

If you set a deadline and tell everyone you know about it, with one small motion you'll start a silent timer. A doomsday clock of sorts. The pressure will build, but in the end you will push something out the door.

With Workflows we found a few extra tidbits we wanted to add before launch. We also realized we had overlooked re-working our onboarding flow to match the new stuff we were launching… but we didn't push the date.

Because we'd committed to it in public.

But Beware…

There's a balance here. Most of us are building companies because we want to enjoy the journey. So if you're working yourself to death because you're constantly setting overly ambitious deadlines, you're doing it wrong.

If you go on one too many death marches due to unrealistic deadlines you'll reach *deadline fatigue*, where they don't motivate you any longer. Don't do this.

I think it's good practice to put your feet to the fire and commit to a hard deadline a few times per year.

For me, the two weeks after we set our Workflows deadline were the most productive I've had in months. Something about setting that deadline (let's be honest, it was the fear of missing it) pushed me to become the highly-focused, insanely productive Rob I catch a glimpse of now and then.

So if you find you're in a rut, not making progress, trying to figure out what to polish next, or you haven't shipped anything in months…set a deadline. In public. And see how it makes you feel.

Hopefully, it scares the hell out of you. That's how you know it's working.

IT'S EASY TO BE GREAT... IT'S HARD TO BE CONSISTENT[16]

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 To-

16 Original location: http://robwalling.com/2010/11/04/its-easyto-be-great-its-hard-to-be-consistent/

night 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 Kutcher 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 un-

quenchable, 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 is: *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, and 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.

LESSER KNOWN TRAITS OF SUCCESSFUL FOUNDERS[17]

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,

17 Original location: http://robwalling.com/2010/09/16/lesserknown-traits-of-successful-founders/

and make a swift but logical decision to move forward or cut bait, the 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 *alwayshave* 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.

THE TERROR OF FIRSTS[18]

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, were my first:

- Face-to-face with a potential client
- Published article
- Blog post
- Product launch
- Product acquisition
- Time speaking at a user group
- 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

---

18 Original location: http://robwalling.com/2010/08/05/why-startup-founders-should-stop-reading-businessbooks/

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%?

WHY STARTUP FOUNDERS SHOULD STOP READING BUSINESS BOOKS[19]

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.)

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

19 Original location: http://robwalling.com/2010/08/05/why-startup-founders-should-stop-reading-businessbooks/

– 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 Procter & 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 takeaways 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.

ONE OF THE MOST TIME-CONSUMING STARTUP ROADBLOCKS[20]

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

---

20 Original location: https://robwalling.com/2009/07/14/one-of-the-most-common-startup-roadblocks/

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 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 al-

most 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 Upwork.

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 MicroConf).

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?

DON'T PLAN TO GET RICH FROM YOUR STARTUP[21]

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 are better at marketing software have built products and still work for a living.

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 hun-

---

21 Original location: http://www.robwalling.com/2010/06/10/dont-plan-to-get-rich-from-your-startup/

dreds 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 thing 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 book about starting a startup is titled *Start Small, Stay Small*)

HOW TO AVOID THE THREE STARTUP DANGER POINTS[22]

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 creation 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 overthink 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.

---

22 Original location https://robwalling.com/2010/05/21/how-to-avoid-three-startup-danger-points/

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.

While you'll never remove all uncertainty, you can remove enough that making the niche decision is a bit more clear-cut than the dartboard 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, committed 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 the construction of your sales website, or the HTML/CSS conversion, or copywriting can remove 20-80 hours from your timeline 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 start-up founder.

No, the first few months after launch is 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 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.

An entrepreneur 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.

THE SOFTWARE PRODUCT MYTH[23]

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

23 Original location http://www.robwalling.com/2008/11/18/the-software-product-myth/

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 greenfield 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 boneyard 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.

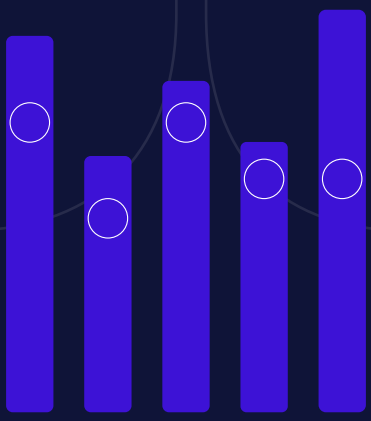
But there is a *dip* before you get to this place of exponential leverage and proverbial cash. A big dip. And if you can get through it once, it's more likely that you'll be able to get through it with your next product. And the one after that.

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.

03

RUNNING YOUR COMPANY



FROM 160 HOURS TO 10: A TALE OF AGILE BUSINESS PRACTICES[24]

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

24 Original location: https://robwalling.com/2010/04/07/from-160-hours-to-10-hours-a-tale-of-agile-business-practices/

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 of 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 of 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 an 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 we don't want to launch a semi-functioning solution because we'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 an entrepreneur 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 an entrepreneur:

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

HOW TO DETECT A TOXIC CUSTOMER[25]

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 pro-*

25 Original location: https://robwalling.com/2010/12/09/how-to-detect-a-toxic-customer/

gramming 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, 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 voicemail 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 whomever 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.

WHY YOU SHOULD BUILD YOUR STARTUP TO SELL (BUT NOT TO FLIP)[26]

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

---

26 Original location: https://robwalling.com/2010/07/01/why-you-should-build-your-startup-to-sell-but-not-to-flip/

its value based on factors the market values, since these factors will make it a better company to own in 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:

1. Repeatable
2. Teachable
3. 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 inter-

vention, 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 business 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.

YOU CAN'T MAKE MONEY CHARGING $1 PER MONTH[27]

I recently received the following question from a reader:

"[When we spoke at a recent conference] I had been thinking of a $1/month price point [for my product aimed at teachers] to make it a "no-brainer," and you strongly advised against it, suggesting $5-7 at a minimum. Are you concerned at all about the fact that teachers could continue using the existing workaround solution? I wonder if I provide enough value to rationalize $5-7. Maybe it would be better for me to find more ways to add value rather than lower my price point?"

My Response
Trying to make money selling an app for $1/month is crazy unless your market is gigantic and you have the expertise or the funds to reach them (and even then, support will kill you).

Let's look at some numbers:

1. If your goal is a meager $2k per month you need 2k customers.

---

27 Original location: https://robwalling.com/2011/06/27/you-cant-make-money-charging-1-per-month/

2. To begin, that's a lot of non-technical customers to support for that little money. You'll still be working a full-time job at that point so it'll be nights and weekends. Not cool.

3. To get 2k customers with a 1% conversion rate you'll need 200k unique visitors (total, not monthly). If you crank hard on promotion and word of mouth you'll ramp up to thousands of uniques per month. Unless this is a massive market where you can rank high in a major search engine, it's going to take you years to drive 200k uniques since you don't have the budget to run ads.

4. Speaking of ads…at $1/mo (or $7/mo), you won't have the budget to run ads. The cost to acquire each customer (CAC) will be too high compared to the lifetime value (LTV) you will receive from each one. So you will be forced into "free" sources of sign-ups like SEO, word of mouth and vitality. This is not the end of the world, but if you don't know how to optimize those super-competitive channels, it's not going to be easy.

5. In general, and counterintuitively, lower-priced products tend to have higher churn, higher support burdens, and more price-sensitive customers.

6. Since running into this issue myself with several B2C plays I have personally moved each successive product upmarket, and moved to selling to businesses. Even at $10/mo, it's hard to grow a SaaS past tens of thousands per month in MRR without a lot of incoming traffic. If that's your end goal, awesome! If you're early in your journey

7. But honestly, I would try what I call "Aspirational Pricing," which is when people ask you to lower your price, you ask, "What would I need to build to make it worth the price I'm charging today?" And then build those features (as long as they are within your product vision).

8. Charging $50 or $100/month (or more) is such a better way to go, in my experience with my own companies and those I advise.

9. Also, in this case, if you are stuck charging $5 or $7/month, I would charge annually so you're at least getting the cash up-front to allow you to grow, re-invest into the product, and grow the business faster.

FOUR THINGS I LEARNED BY ASKING MY CUSTOMERS[28]

PMF Survey is a tool that helps you edge into customer development. It's a free tool put together by Sean Ellis as a way for product owners to easily survey their customers using pre-written questions.

You enter your product name, and voila – 8 questions that Sean Ellis has used many times to achieve his massive successes with companies like DropBox, Xobni, LogMeIn, and Lookout.

I've had this on my to-do list for DotNetInvoice since PMF Survey launched, but I wanted to wait until our QuickBooks integration launched, which finally happened last month. I was further spurred into action at MicroConf; one of the key takeaways from the conference was that I needed to be talking more to my customers.

So two weeks ago, I dove in head first and emailed the survey to all of our customers. The results were surprising…

The survey consists of 8 questions covering how people found out about DotNetInvoice, how they would feel if they could no

28 Original location: https://robwalling.com/2011/07/15/four-things-i-learned-from-asking-my-customers/

longer use it, what they would use as an alternative if DotNetInvoice were no longer available, and several others.

The idea behind the survey is two-fold: to determine if you have a core group of users who really need your product, and then find out the most important features for them, what other features they want, etc…

The problem with traditional surveys asking "What should we build next?" is you might get 40 feature suggestions, but you have no idea which ones are going to cater to your core group of customers, and thus hopefully result in more sales to customers like them who really need your product.

I'm not going to run through the details of every question but rather share some insights I took away from the survey and how it will shape the future of DotNetInvoice. Keep in mind these are specific to DotNetInvoice and may or may not apply to your product (I recommend using PMF Survey to find out for yourself).

Insight #1: SEO Traffic Converts Extremely Well
Of the people who responded to the survey, 85% of people originally found us through Google. Since the survey was only sent to existing customers, this number blew my mind. Search engines account for less than 40% of our overall traffic, but they appear to have a much larger impact on sales.

I should know this information. I have Google Analytics set up to track goals. But Analytics tells me that only 37% of our sales come from search engine traffic. But I've always known that Google Analytics is a victim of the "last click problem" whereby it assigns the conversion to the last thing a user clicked, rather than how they originally found out about your product.

But I didn't realize it was off by this much.

I attempted to use KISSmetrics last year to avoid the last-click

problem, but I ran into problems with the tracking. But the first thing I'm going to do is get it going again to figure out which of these numbers is correct.

If it's really 85% I know where I'm investing my time over the next 6-12 months.

Insight #2: We Appear to Have Product-Market Fit
According to Sean Ellis, there is a good chance you have product-market fit if more than 40% of customers would be very disappointed if they could no longer use your product. We are at 48%, followed by 33% who would be somewhat disappointed.

While I would like the "very disappointed" number to be in at least the 60% range, I do find some salvation in the fact that we seem to have found a core group of users who really need our application.

It also appears that the majority of "very disappointed" respondents operate within one or two related verticals. This is great news not only for the future of our marketing, but for our product development. We're now looking to focus our efforts on specific features that should increase sales in these two niches, even at the expense of sales in others.

Insight #3: We Have Competition I Didn't Know Existed
We've been aware of Freshbooks for years, and 37% of respondents indicated if they could no longer use DotNetInvoice that they would go there. However, the shocking stats are as follows:

- 26% of respondents said they would write their own if DotNetInvoice were no longer available

- 22% said they would use no alternative at all, which I think means MS Excel or MS Word

The key action item here is to update our marketing messages to

properly capitalize on our previously unknown competitors.
Our home page will soon include verbiage along the lines of: "Stop using Word and Excel, save time and get paid faster with DotNetInvoice."

The headline on our developer landing page will likely change to: "Don't start from scratch, start with DotNetInvoice."

Insight #4: Our Primary Benefit is Not What I'd Thought
For years, I've thought our #1 competitor was Freshbooks, and so have geared some of our marketing on our key advantages over them: you stay in control of your data, and source code is included so you can customize as much as you need.

However, it had not occurred to me that the following would come back as our primary benefits:

1. Automated invoices and reminders

2. Easy/Simple to use

Perhaps retaining control of your data and having source code is an expected feature for our customers, so they didn't bother to mention it? Because while we currently tout automated invoicing and our simplicity, these are not our #1 marketing message, since the competition essentially has both of these features as well.

So I don't expect to make any changes to our messaging based on this realization, but I am pleased to now have a mental list of the top 4 reasons people are most likely to buy DotNetInvoice.

Conclusion

For 15 minutes of setup and 90 minutes parsing results, this was one of the most fruitful 105 minutes I've spent in some time. The findings above, along with a few others are going to shape the direction of DotNetInvoice for the next year or more.

04

RECRUITING & RETAINING DEVELOPERS

HOW TO RECRUIT A DEVELOPER ENTREPRENEUR FOR YOUR STARTUP [29]

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 an 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*

29 Original location: https://robwalling.com/2008/03/13/how-to-recruit-a-developer-entrepreneur-for-your-startup/

> *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 a 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[30]

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

30 Original location: https://robwalling.com/2007/04/09/how-to-attract-software-developers-fit-your-startup/

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 rewards, and the prospect of building something really cool in a short amount of time that could make a difference in people's 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 into 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 rewards. 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 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.

PERSONALITY TRAITS OF THE BEST SOFTWARE DEVELOPERS[31]

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?

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 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.

---

31 Original location: https://robwalling.com/2006/08/20/personality-traits-of-the-best-software-developers/

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's software 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 of 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 its 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 rebooted. 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 2 am 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 an 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.*

NINE THINGS DEVELOPERS WANT MORE THAN MONEY[32]

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?

---

32 Original location: https://robwalling.com/2006/10/31/nine-things-developers-want-more-than-money/

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 sin-

gle-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:

Criteria #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 a 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.

Criteria #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.

Criteria #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 program-

mers 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.

Criteria #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.

Criteria #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.

Criteria #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.

Criteria #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 underserved financial community.

An Albertson's *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.

Criteria #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*.

Criteria #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 suck 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.

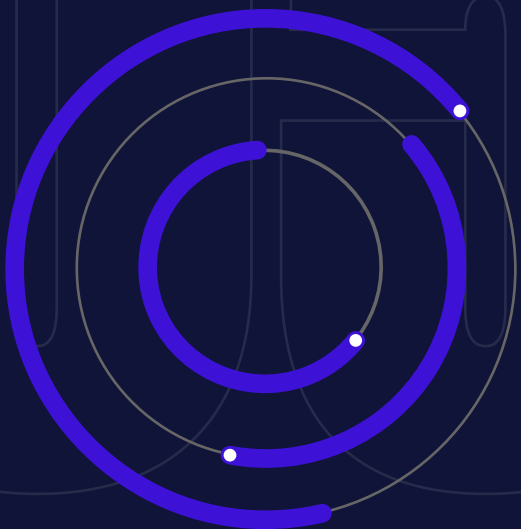
Wrap Up

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?

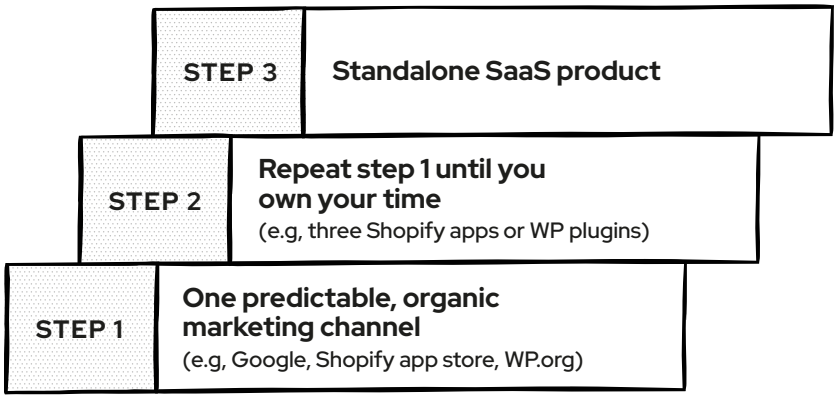
If the answer is "a long time ago" then you have some work to do. Send this article to a few of your colleagues and *start discussing how to enact change.*

05

ENTREPRENEURSHIP



THE STAIR STEP METHOD OF BOOTSTRAPPING[33]



Between this blog, my podcast, MicroConf and TinySeed, I've had the privilege of watching hundreds of entrepreneurs launch products over the past decade (even into the thousands, depending on how you count).

After a while, I started to notice a pattern emerging among the pool of bootstrappers who were able to successfully replace their income, buy back their time, and quit their jobs.

---

33 Original location: https://robwalling.com/2015/03/26/the-stair-step-method-of-bootstrapping/

I first captured this pattern in my notebook in early 2009, but I didn't have enough data to be confident in the theory. I started embracing it as a framework on my podcast around 2010, and in 2013 I gave it a name.

I call it The Stair Step Approach. After fully fleshing it out in my talks at DCBKK and MicroConf Europe last fall, I've been struck repeatedly by how many successful bootstrappers have followed, or are currently following, this trajectory.

The interesting part is that my own path moving from consulting to products followed the same steps, as you can see in my product revenue chart from the past decade:

Each revenue jump is when I made the move to the next step of the Stair Step Approach.

I decided it was time to put pen to paper and lay out what I see as a repeatable path with a higher-than-normal success rate, to bootstrapping yourself to the point of quitting your job.

Here's how it works:

Step #1: Your First Product
Everyone knows how difficult it can be to launch your first product, and frankly, a lot of the startup advice out there just makes it harder. From validating an idea, to testing message-to-market match, coming up with a marketing plan, and actually building the product, it's a miracle anything gets launched at all.

With that in mind, how do you give yourself the best chance of avoiding failure?

In my experience, the biggest pitfall that trips up first-time product people is trying to create something too complex.

The strategy that seems to give people the best chance of success is creating a simple product, with a simple marketing plan – one that only requires a single traffic channel.

Specifically, I'm suggesting that you don't get started by building a stand-alone subscription software (SaaS) product. Recurring revenue is the holy grail for bootstrappers, as we'll discuss later in this essay, but it also makes your business more complex if you try to launch a stand-alone product that you have to not only build, but market on your own.

It's much easier to sell an add-on to an existing ecosystem like a WordPress plugin, a Shopify app, a Heroku add-on – they're usually small things to build, relatively inexpensive for customers to purchase, and you have built-in discovery through the plugin repository or app store.

Other examples include Magento or Drupal add-ons, a Photoshop plugin, a WordPress theme, or an ebook or course. There's an extensive list of these app stores/marketplaces over on the Rocket Gems blog.

These smaller projects might not be as "sexy" as trying to disrupt marketing automation, and they probably won't land you on TechCrunch, but they will help you become profitable much, much earlier.

A project like my last SaaS app, Drip, is a serious undertaking. I'm confident I would not have been able to make it work 13 years ago when I first started building products.

As for a single traffic source, focus on figuring out one way of generating customers, instead of trying to master all the different traffic channels at once.

Getting those initial early sales is a big win.

In my case, that meant getting good at SEO with my first project, instead of trying to learn SEO, Adwords, Facebook ads, etc. all at once. I've seen many other people focus specifically on generating free downloads from WordPress.org and then up-selling premium plugins or ranking for appropriate keywords in the Shopify or Heroku app stores.

This focus will help you start generating revenue without worrying about becoming a master marketer, and let you develop new skills one-by-one as you need them.

Step #2: Own Your Time

Step 2 is doubling down on the model that worked in Step 1 and repeating it until you own your time. You may be able to grow a single product to achieve step 2, or you may need to cobble a few together.

In essence, once you've launched your first successful product and it's making a bit of money, try to understand if you can grow it to that "full-time income" level, or whether you need to repeat the process and launch a second or third product in order to buy out your time.

One of the biggest mistakes I see founders make is abandoning what already works and trying to take on a bigger challenge before they're making enough money to own all of their time in a given week.

Obviously, most WordPress plugins won't pay you enough to quit your job, but three might – that's exactly what worked for Dave Rodenbaugh. Once he figured out how to successfully sell premium add-ons to users of his free plugin, he developed two more plugins with premium options, instead of jumping to SaaS or trying to figure out how to promote his plugins with Facebook.

And now Dave has moved on to his next product, a very successful "step 3" SaaS application called Recapture.

Phil Derksen is another example with 2 WordPress plugins: Pinterest Pin it Pro and Stripe Checkout Pro for WP. He quit his job a few years ago by stacking their combined revenue into a full-time income.

Richard Chen of PHP Grid recently made the jump from one-time sales to leaving his job, and is working on climbing to the next step.

David from FatCatApps is doing great with his easy pricing tables and easy opt-in plugins.

And Fri Davies from the Dynamite Circle has done this with his Magento add-ons.

When I first started moving out of consulting in 2005, I acquired a product called DotNetInvoice, a straightforward piece of invoicing software that I promoted almost exclusively through SEO.

The revenue from that product grew incrementally until 2008 when I started acquiring and building more products instead of trying to optimize what I was already working on.

Those were still simple projects, like a job board for electrical linemen, an e-commerce store for beach towels, and a couple of ebooks – all one-time sales promoted that relied on SEO traffic or AdWords to make sales.

So why did that grow my business faster than testing and tweaking each of these products individually?

Lifetime value. The lifetime value (LTV) of nearly all of these products was not enough to warrant anything except "free" traffic from an organic source like Google or the WordPress plugin repository.

But I didn't know that at that time, which lead to me wasting more than a year trying to scale my beach towel store. But when you're selling something that has an LTV of $10-15 dollars, you just can't make paid advertising work. Or content marketing. Or any of the other approaches I would use to grow an app with a higher LTV ($150+).

So in Steps 1 and 2, I see people succeeding by sticking to free traffic channels (typically organic or viral), sticking to the first one that works for them, growing it until it plateaus, and focusing on replacing your income with the first product, or by repeating it with multiple products.

None of my early projects were particularly glamorous, but when I stacked them together I built enough product income to buy out 100% of my time and quit consulting.

That let me move on to step 3, where I was able to start focusing on bigger projects, with higher LTVs that gave me room to experiment.

Step #3: Recurring Revenue

Now that you're generating enough income to justify going out on your own, and you have the experience and the mindset of

someone with a few successes under their belt, it's time to level up and take a bigger risk by going after a stand-alone product with recurring revenue.

Recurring sales are the holy grail for bootstrappers for a reason – instead of repeating the painstaking process of selling new product month after month, every customer increases revenue for this month, next month, and beyond.

While some percentage of your customers will churn, this model gives you incredible leverage to grow your business – if you're providing a good product, you will retain the vast majority of your sales month over month.

That's a big part of the reason why SaaS is so popular with bootstrappers.

But the other side of the SaaS coin is that there's along ramp to any kind of substantial revenue, which is why I recommend you complete steps 1 and 2 before moving to this step.

Aside from the fact that recurring revenue tends to yield much higher lifetime values – think about how much you've paid Basecamp or your hosting company over time – the high dollar value that a new customer is worth to your company opens up a world of new marketing approaches you can attempt, such as PPC, display ads, content marketing, integration marketing, etc.

When I bought HitTail and optimized its sales funnel, it more than doubled my annual revenue, even though the average purchase per customer was barely in the double digits.

But because it's relatively inexpensive on a monthly basis, customers stay with us long enough to create a nice LTV which gave me the flexibility in ad spending to figure out a profitable Facebook ad campaign that brought more than a thousand new customers over the course of about 6 months.

That kind of growth is the promise that drives a lot of new founders to dive headfirst into SaaS projects. But frankly, I hope this article has made you reconsider that it's probably the wrong place to start due to the technical and marketing complexities (not to mention the competition) that go along with SaaS.

I don't think HitTail or Drip would have taken off if they were my first projects – I don't know that I would have had the time, money, skills, or confidence to give them what they needed to succeed.

Moonshots

The tech media likes to focus on moonshots because they're exciting, but their focus on the 1-in-10,000 that work means they ignore the hoards of people who have tried and failed because they try to play in the NFL before they've learned basic blocking and tackling.

There is no guarantee that The Stair Step Approach will help you launch a successful product right out of the gate, but based on patterns I'm seeing in the bootstrapper community it's a nice, low-risk path to a successful software company.

SHOULD YOU BUILD OR BUY YOUR STARTUP?[34]

Startups. 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.

---

34 Original location: https://robwalling.com/2008/10/03/should-you-build-or-buy-your-software-product/

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 software 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 startup 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 startup.

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 startup 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.

In 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 startup 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.

THE INSIDE STORY OF A SMALL SOFTWARE ACQUISITION[35] (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

35 Original location: https://robwalling.com/2007/09/16/inside-story-small-software-acquisition-1-of-3/

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.[36]

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

36 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.

building a product once, and then sell it to many customers.

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.

THE INSIDE STORY OF A SMALL SOFTWARE ACQUISITION[37] (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 screenshots 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 makeup DotNetInvoice (then at version 2.0) were mine.

---

37 Original location: https://robwalling.com/2007/10/10/inside-story-small-software-acquisition-2-of-3/

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 documents 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 toward experienced .NET developers. The product now includes a PDF with screenshots that walk you through the setup 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 rolled in. Things were looking up.

Until the end of the first month rolled around.

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 floodgates 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.

THE INSIDE STORY OF A SMALL STARTUP ACQUISITION[38] (PART 1 OF 3)

Based on the title of this essay you might be thinking I have mad stacks of money in the bank.

That I've had a few "exits" and instead of hunkering down and writing code for 6 months I opted to talk to a few of my buddies at the yacht club and purchase a primed and growing social network for somewhere in the mid-seven figures.

Indeed, I did buy my latest startup, but the deal was done from a spare bedroom of my suburban home in Fresno, California for less than most people pay for a new car. And the funds came from revenue generated by my portfolio of web applications and websites that I've built over the past several years.

This acquisition is a long story, but if you have a few minutes let me tell you the best parts.

---

38 Original location: https://robwalling.com/2012/01/25/the-inside-story-of-a-small-startup-acquisition-part-1/

My Background

If you haven't followed my blog in the past, I'm a guy who launched several web startup/product ideas, acquired several more, and had many failures and enough successes that I was able to shut down my consulting firm back in 2008.

My apps range from an SEO keyword tool, an ASP.NET invoicing system, a web application for creating wedding websites, and a handful of others.

I'm all about figuring out what's going to make a person happy and then going after that with ravenous determination, instead of pursuing what we're told is going to make us happy by the tech press (raise funding! exit big! lose control of your company and get fired by the board!).

So I tend to focus on ideas that have a 1000x higher chance of success than the next un-monetizable social website you have in mind, but the success I strive for is a bit more modest. Probably close to 1/1000th of the payout of a big exit.

But I believe this approach is far more likely to make you happy, and far more likely to actually make a difference in the lives of more than the handful of people who hit the startup lottery each year.

I believe this so much that I've written a manifesto and a book and hundreds of blog posts on the topic and hold a conference every spring that focuses on self-funding your startup. I've put all my eggs in one basket, and that basket and the eggs were bought with money from my own self-funded pockets.

In Search Of…

So that's an intro to my startup philosophy, but where was I? Aaaah yes…the search.

After co-hosting MicroConf in early 2011 I was on the prowl for the

next big thing. After a few weeks of soul searching I determined I was going to build a SaaS app. Another week left me with a short list of problems that entrepreneurs need help with (since I'm pretty familiar with the space), and after another few days had them ranked in order of my interest.

In the top 3 was the following problem: entrepreneurs need more organic search traffic to their websites. That phrase may sound exciting to you, or it may be a complete snooze-fest.

The bottom line is that many successful startups (far more than you hear about) are masters at SEO. People don't tend to talk about SEO as a sexy marketing approach, but it can generate enormous amounts of highly-targeted, high-converting traffic.

Depending on your market, the ROI can be better any other traffic source you can find (with viral traffic the most common exception).

As I pondered the entrepreneur SEO question I realized there was an application I'd been using for years that pretty much fit into everything I mentioned above, and it was all but abandoned. Imagine the luck!

HitTail is a tool that tells you the most promising organic search terms you should target based on your existing traffic. It has an algorithm that analyzes your visitor stream in real time and provides you with a simple list of precisely which keywords you should be targeting to maximize your organic search growth.

And I'd been a user since 2006. Even through the five-day outage in early 2011, and the semi-regular downtime throughout 2010 and 2011.

Customers were bailing on the service because of the frequent downtime. But there was a hard-core customer base that had been around for years and had stuck with the service because

the main algorithm that provided recommendations had never stopped working, even though everything around it had crashed and burned.

Closing

On a warm day in early June I cold emailed the owner through the website contact form. I have to admit – I didn't know if I would get a response at all. But within 24 hours we were off and running, discussing a potential acquisition.

The site was once great; marketed by a high-end NYC PR firm. But when the firm had focused its attention on other things the site was neglected, and the technical headaches ballooned as the original developers left the company. Needless to say, the owner was definitely interested in discussing an acquisition.

So I made my offer and she countered with 5x the amount. Choke! I've acquired many web apps in the past, and I wasn't prepared to pay "dot com" pricing for it. Although it had been mentioned in Inc Magazine, BusinessWeek, The Wall Street Journal, PC World, and many other mainstream news outlets, given what I know about rehabbing applications I had a firm figure in mind.

The site generated revenue but it was losing customers and the hosting bill was hefty given the performance requirements of the site (it's basically performing real-time analytics).

It took three months of emails before we settled on the final terms. The close was swift: I had a lawyer draw up an agreement, and we used an escrow service to handle the money transfer. It was a clean deal as deals go, and at the end of August, 2011 I was the new owner of HitTail.

And I was running scared…

Step #1: Like a Chicken With No Head

The big issue is that the server hardware was sketchy at best. It

had ancient hard drives, an installation of Windows Server 2003 that hadn't been refreshed for 5 or 6 years, and two SQL Server databases that had not been re-indexed (or had any visible maintenance) for 3 years.

All the while continuing to handle 10-30 DB inserts per second. Every day. For years. There are literally over a billion rows in the database and no one had touched it since 2008.

So I scrambled. I found an awesome DBA on oDesk and moved the 250GB database to a new cloud server infrastructure. A few weeks of prep and an all-nighter later I was feeling much better about HitTail's stability.

Step #2: The Funnel

The next step was plugging the funnel. To give you an idea of the state of the site, here's a glimpse of the home page from a month ago (and I think this is how it had looked since 2006):

In addition to design issues, there were major leaks in the sales funnel. Examples include:

- Many 60-day trials had never ended. There was no script running that ended them so people continued to use the site for free for years.

- During the trial period a user received exactly zero emails. No reminders to check out their fancy new suggestions. No inquiry as to why they never installed the code. Zip.

- The website was architected such that it was easy to just wander off into the sunset and never be heard from again. No effort had been put into moving people from content pages back into the main marketing website that promoted the benefits of using the product.

Beyond that, there were broken features. Oy vey, were there broken features. I commented out 3 or 4 "top line" features that were such a big deal they appeared on the pricing plan page. But no one had used them in years because they didn't work!

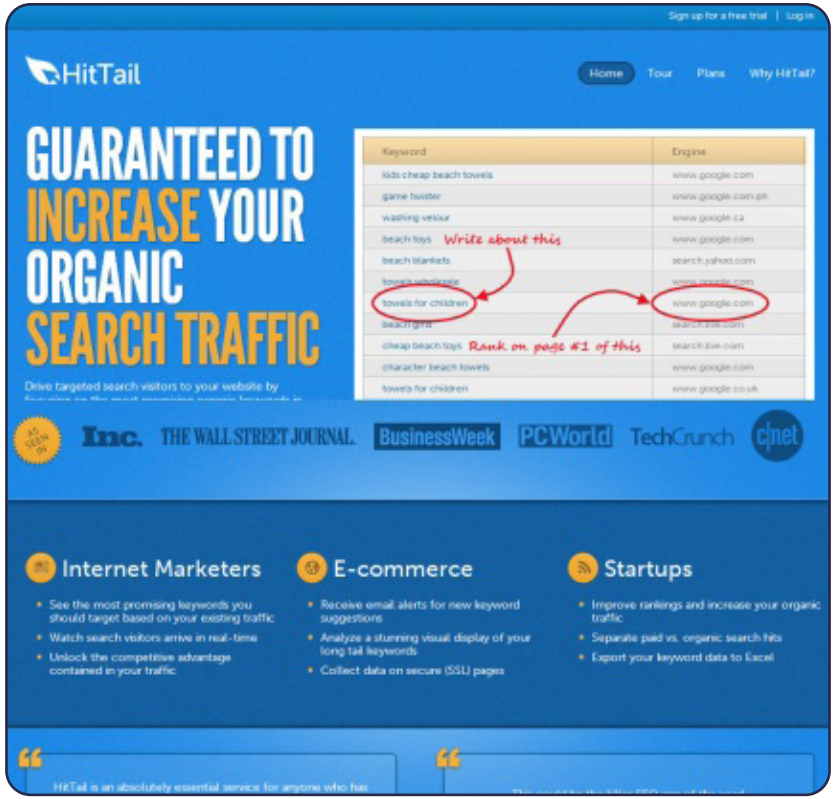
And the algorithm that made suggestions was operating at around 10-20% effectiveness. In other words, due to the way Google had changed the way it passed URLs in the query string, the algorithm was missing a slew of valuable keyword suggestions.

I can almost hear the voice in your head: *"Something must be done!"*

And that something involved me hiring a developer and a designer to help me revamp this thing (both contractors, of course, since I don't hire employees). I also put in quite a bit of coding time myself.

And three months later, the joyous visage you see below you was unveiled upon the world (this is the new HitTail home page, re-

placing the one I showed above):



'll be the first to admit: it's blue. But I like it. And so do people who need organic keyword suggestions, apparently, because it's converting very well so far.

Conclusion

Those are the highlight of how, 4 months after acquiring a dying web property, I brought it back from the dead with the help of my ragtag team of intrepid oDesk contractors.

And you may be wondering why I bought a startup instead of building it from scratch. Stick around for that; I cover it in part 2. There are far too many details to include in a blog post, so if you like audio and want to hear even more about the acquisition, I

was interviewed on this very subject for 90 minutes on TechZing. Check out episode 165.

THE INSIDE STORY OF A SMALL STARTUP ACQUISITION[39] (PART 2 OF 3)

This is part 2 in a series covering my acquisition of HitTail; part 1 went to the top of Hacker News last week and I have a slew of questions from that discussion that I will answer next.

But first I want to address the most common question I hear when I tell someone I acquired a startup:

Why did you buy instead of building?

If you're a developer you're probably scratching your head wondering how I could pass up the chance to do the awesome green field development. A new project with no legacy baggage…this is the stuff we live for!

But I did indeed opt to plunk down my hard earned cash instead of hunkering down for 6 months in my dev cave, and what follows are my reasons for doing so.

---

39 Original location: https://robwalling.com/2012/02/02/the-inside-story-of-a-small-startup-acquisition-part-2/

Reason #1: Saved 12-18 months
I have a little theory that a v1.0 should take 4-6 months from inception to launch. This includes time for marketing (which should start before you write a line of code), planning, development, launch, etc…

And if you have the luxury of working on it full-time then by all means you should get it out the door faster. Almost without exception, the sooner you get in front of real customers the better off you'll be.

But in general, this is "Rob's Theory of Time to 1.0": 4-6 months, 300-600 hours.

But there's another time frame, and I call it the "time to flywheel."

A flywheel is a large, heavy wheel that requires an incredible amount of effort to get moving, but once it's moving it can continue to do so under its own momentum for quite some time, without additional effort. The time it takes to get your business to "flywheel" will vary, but I've found that it's typically 6-12 months after launch.

"Time to flywheel" is not profitability, but an app is typically on a rapid trajectory toward profitability when it hits.

It's also after you've confirmed you've built something people need and are willing to pay money for, and have found your market (some would say product-market fit). You'll also have some marketing channels that are bringing in new customers for less than they pay you during their time as a customer.

This is the point where you have a bit more work to do in order to scale, and when smart founders who are interested in funding begin to seek it out. My typical estimate when asked how long it takes to get there is 12-18 months. Some companies can pull it off in 9, for others it takes 24 if they are ever able to achieve it.

But the sweet spot is around 12-18 months.

Getting to "flywheel" is a slog. It's painful (and yet also exhilarating). And frankly, I'm not a patient person when it comes to growing a business.

Given the hourly rate at which I value my time, and the value I place on 12-18 months of effort, buying an application like HitTail is a ridiculous bargain. It may be the best deal I receive on anything I buy during the next year.

Reason #2: It Has "Good Bones"

There's something to be said for acquiring an app that's already worked out the v1.0 kinks. HitTail has been around since 2006, and although some of the software architecture and code structure leaves something to be desired, a lot of bugs were ironed out in the early days.

Any problems that existed when I took it over had crept in over time due to changing APIs, lack of maintenance, and failed hardware.

In general, the code had been purring away for 3 years without intervention, and that indicated one thing to me: this thing had good bones.

"Good bones" is an expression realtors use about a house to indicate that the major systems like plumbing and electrical are solid, but that the house may need cosmetic upgrades like new paint and flooring. It means the house is pretty solid as it stands, and that with a bit of effort, it could be more attractive and more valuable.

It's an expression I would apply to HitTail, and one of the reasons I decided to acquire it.

Reason #3: Dramatically Reduced Market Risk
When you set out to build and launch a product there are three types of risk: product, market and execution.

- **Product risk** looks at whether or not you'll be able to build the product. These days, most web apps we see have only a small amount of product risk.

- **Market risk** looks at whether there will be people willing to buy your product if you build it. This is the risk factor I encourage founders to look at the most, especially before they write a line of code. Market risk is the deadliest of the three because if you build something for which there is no market, it's nearly impossible to adjust course later and fix it.

- **Execution risk** is whether you can pull off the whole package. Can you bring everything together, build and launch the product, and get it in front of the right prospects?

Given these, market risk is the one that scares me the most. Every time.

I have confidence in my ability to execute, but anything I can do to reduce market risk early is a no-brainer. Buying an app with paying customers is one of the best ways to ensure you have something people will pay for.

Reason #4: Better than Outsourcing
I talk a lot about outsourcing. Last year I paid $100 or more to at least 15 contractors ranging from developers to an infographic designer. And I'm a firm believer that you must learn to outsource or hire employees if you want to grow a business.

But I would take a completed product over a spec every time.

Outsourcing works, but avoiding the possibility of hiring a crappy developer, building the wrong feature set, and the enormous

amount of time you need to invest to manage a development project is typically well worth the purchase price.

Counter-Reason #1: "If you buy, you're not a real founder!"

This is from the internal monologue that attempted to talk me out of the acquisition.

At one point I actually stopped to wonder what people would think of the fact that I didn't build HitTail myself from the ground up.

Are you a real technical founder if you don't write every single line of code for your v1? Or at least oversee every single line of code as it's being written by your team?

I've built and launched apps from the ground up in the past. Am I still a real founder if I skip it this time?

The conclusion I came to is *that it doesn't matter.*

Growing HitTail into a more profitable business is the next phase of my learning (and the next test of my chops). Both personally and career-wise it's the right decision.

Everything else is semantics.

THE INSIDE STORY OF A SMALL STARTUP ACQUISITION[40] (PART 3 OF 3)

This is the final installment of a 3-part series covering my acquisition of HitTail. I'd originally planned on a 2 part series, but when parts 1 and 2 went to the top of Hacker News I received so many questions that I decided to add this prologue to answer them.

Every question below has been asked via email, comment or Twitter.

"***Could you talk a bit about the diligence you did during the acquisition? How long into the discussion before you commenced diligence? What did it entail and how did you verify what was told to you?"***

Due diligence is the most important part of the process because it dictates whether you should move forward with the acquisition, and at what price.

As such, due diligence consumed the majority of the two-month

40 Original location: https://robwalling.com/2012/02/28/the-inside-story-of-a-small-startup-acquisition-part-3/

negotiation period between myself and the seller. I asked for all of the following information:

- Monthly revenue
- Monthly expenses
- Number of paying customers
- Number of free users
- How many customers cancel each month
- Monthly unique visitors
- Number of trial sign-ups per day
- Number of new paid accounts per month

The problem with the list above is that it's long so as a buyer you need to take it slow and not blast the seller with it all at once. Once the seller indicated a sale might be possible I asked for the first 3 from the list above, asked for a few more in a subsequent email, and so on.

In addition, only some of the information above is verifiable, and we did that with screenshots and exports from Google Analytics. At some point, I had to trust the seller, and I gained trust that she was legitimate fairly quickly after learning more about her background, and Skyping with her.

Bottom line: it's pretty easy to come up with a laundry list of things to ask for. The trick is narrowing that list to the point that the seller will tolerate it. I've had several negotiations end with the seller indicating they don't have the time or desire to dig up all of the data I've asked for. At that point, you have to make a decision to reduce your list or shut down negotiations.

The point of the above data is to construct a picture of how the business works, where new customers come from, and calculate a customer lifetime value (LTV). The LTV number was critical for me, as it dictated what kind of marketing approaches I would be able to use and still turn a profit.

Details on which costs you chose to attack first (technical debt vs. free subscriptions vs. UX) is something all web businesses should think about.

HitTail had some stability issues when I acquired it – it was running on old hardware and drives would fail every so often. So my priorities from the start were in the following order:

1. Stabilization

2. Plugging the marketing funnel

3. Spreading the word

Priority #1: Stabilization
Stabilization involved moving all code and the 250GB database to a new server to ensure high uptime and improve performance. This step took was more involved than I had originally anticipated, but with the help of a contract DBA, we planned, prepped, and moved during the first 3 weeks I owned it.

During this time I was also fixing bugs like crazy: I fixed or removed around 25 broken features from the app. Developers in the audience will cringe, but for speed's sake and because the app was already limping along, I made all changes live on the server in the production environment. I'm thankful to be past that stage.

In fact, the day we were scheduled to move to the new server one of the existing servers failed about 8 hours before the planned move. No data was lost, but it really was just in the nick of time. All of the trouble I'd had sleeping for the past three weeks was

true – these servers were just barely hanging on.

Priority #2: Plugging the Marketing Funnel
Once the app was stable and the major bugs fixed, there was work to do on the marketing funnel. Essentially, the website was in bad shape both visually and in terms of conversions. Most visitors arrived at the site and wandered around for a while, then left. There were trial sign-ups coming in, but not as many as I would have expected based on the traffic levels.

So I began a complete re-design of the site with a high-end designer named Ryan Scherf, and I worked with a contract developer to have the new design applied to the existing 200 or so pages. Then I went through them by hand and added calls to action to the most popular pages. The problem was that people arrived at the site, read an article, and never came to the "core" of the site to learn more about HitTail. I added prominent calls to action on the most visited pages to funnel people into the sales flow.

Finally, I added scheduled tasks to disable trial accounts after 30 days (before this, you could keep a trial account virtually forever), and to email trial users during their trial period, checking in if they haven't installed their code, and updating them on their account status as the end of the trial approached.

This effort raised several key metrics, such as the percentage of trial accounts that install the tracking code – it went from 25% to around 75% – as well as the percentage of visitors who sign up for a trial in the first place.

Priority #3: Spreading the Word
This is the marketing effort I began around 6 weeks ago based on the 11-page marketing plan I've put together over the past 6 months. This effort is still very much in progress; more to come in future posts.

If you can go over the valuation methods each side used, that

would be very helpful to know, even if you don't share the cash figures involved.

I'm not sure what method the seller used – I would imagine I would use the "get as much as you can" method. But my initial offer was based on a multiple of monthly net profit – in the 18-24 month range. Typically I would stick in the 8-14 month range.

I offered a bit more than I typically would given the scope, reputation, and press mentions of HitTail. The seller came back at 5x that amount, and we settled on 1.5x my initial offer and I offered to cover legal fees (around $1500).

I would love more information about the acquisition process. The order of events was pretty straightforward:

1. I send a cold email to the seller (and assume I will never hear back)
2. I receive a reply from the seller
3. We begin some email chit-chat to find out more about each other
4. I ask for some numbers
5. She asks me to sign an NDA, which I do
6. I receive the numbers, and over 2 or 3 more emails ask for additional info and clarifications
7. Based on these numbers, I make an initial offer
8. We email back and forth for 4-5 weeks of price negotiations
9. We settle on a price

10. I contact my lawyer to write up a contract and patent transfer doc

11. We sign everything

12. I open escrow on escrow.com and wired the funds to the escrow account

13. She provides server logins and a slew of additional info, and puts the domain into escrow

14. I take 3-4 days to verify everything, then approve the deal at escrow.com

15. Funds are released to the seller and the domain transfer is completed by escrow.com

16. I stay up until 2 am several nights in a row fixing bugs and stressing that the servers are going to die

I'd love to see something about the mechanics of executing the deals without getting burned. Stuff like finding contracts, escrow, etc…

This depends pretty wildly, but in the past, I've used the Flippa website sales contract, as well as a template I found online (which is not a good idea). Given the size of this one and the complexity of the assets (did I mention there was hardware, software, and a patent application involved?) I hired a lawyer with experience in IP website acquisitions to draft up a custom agreement.

For escrow, I used escrow.com. The site looks dated, but they are the only site I use for this kind of thing.

How did you find HitTail?

I've been a customer for 6 years and saw the service having trouble. It fit my criteria of a SaaS application with a dedicated fan

base (some might say product/market fit) in need of technical and marketing work, and given the LTV, I knew there were many avenues I could use to market it profitably.

Can you say how long you estimated it would take, given your purchase costs and ongoing expenses, to make the acquisition profitable?

I typically shoot for 12 months and, more often than not, break even before that (not including my time investment). My best acquisition in terms of payback was around four months.

For HitTail I did not map out a specific schedule, but it looked like best case, hitting my growth numbers, I would pay back total expenses (excluding my time) in 9-12 months. The worst case was 24 months.

It's too early to tell how that's going to pan out, but given that I'm five months in and just finished the revamp, I'm pretty sure I'm not going to make it back in 12 months.

How did you make the numbers work? As I understand it they had a site that they were putting virtually zero effort into, so they must have wanted a pretty hefty multiple of the annual income. Even if that income was declining, it was still taking zero effort to generate.

It wasn't zero effort – the seller was handling ongoing email requests for support, trying to get new customers set up, and hating life when the site would go down. The site was not on auto-pilot, and although there was no new development or marketing, it required ongoing time and headache to keep it around.

You, on the other hand, had to have assumed you could make the revenue grow significantly via active investment, right? You did mention that [some bugs were] leaving a ton of data and/or money on the table, but you couldn't have known that

ahead of time, could you?

Judging by the existing traffic, conversion rates and how poorly the funnel appeared to be functioning, I had a solid idea I would be able to incrementally improve many steps of the funnel, such as:

- Making the site more appealing and easier to navigate
- Providing a clear call to action on every page
- Providing a clear path to purchase
- Changing the sign-up form from 15 fields to 6
- Asking for credit card up-front
- Offering free tracking code installation
- Touching base with trial users during their trial
- Increasing traffic

Each of the above improves a small step of the funnel, but together they've had a notable impact on the number of new sign-ups.

Perhaps you could explain how you came to your opening offer — did you literally email them and say hi, I'd like to buy this and here is my estimated offer, or did you get some numbers first?

I emailed an introduction and attempted to build credibility that I was a serious acquirer. The seller has to know you are serious or they won't spend time pulling numbers together (or divulging their proprietary data).

Once she expressed interest in a potential sale, I signed an NDA, and she provided numbers. Only then did I tender an offer.

HOW A SEEMINGLY WELL-PLANNED SERVER MOVE CRASHED, BURNED, AND ROSE FROM THE ASHES[41]

In 2011, I acquired a small startup called HitTail. You can read more about the acquisition in the previous essays.

When the deal closed, the app was in bad shape. Within three weeks I had to move the entire operation, including a large database, to new servers. This required my first all-nighter in a while. Here is an excerpt of an email I sent to a friend the morning of September 16, 2011, at 6:47 am:

> *Subject: My First All-Nighter in Years*
>
> *Wow, am I tired. Worst part is my kids are going to be up in the next half hour. This is going to hurt.*
>
> *But HitTail is on a new server and it seems to be running really well. Feels great to have it within my control. There are still a couple pieces left on the old server, but they are less important and I'll have them moved within a week.*
>
> *I'll write again in a few hours with the whole story. It's in-*

41 Original location: https://robwalling.com/2012/06/06/how-a-seemingly-well-planned-server-move-crashed-and-burned-then-rose-from-the-ashes/

sane how many things went wrong.

What follows is the tale of that long night…

The Setup
I acquired HitTail in late August and it was in bad shape. I had three priorities that I wanted to hit hard and fast, in the following order:

1. **Stability:** Before I acquired it, the site went down every few months, sometimes for several days on end. To stabilize it, I needed to move it to new servers, fix a few major bugs, and move 250GB of data (1.2 billion rows).

2. **Plug the Funnel:** Conversion rates were terrible; I needed to look at each step of the process and figure out how to improve the percentage of people making it to the next step.

3. **Spread the Word:** Market it.

The first order of business was to move to new servers, which involved overnighting 250GB of data on a USB drive to the new hosting facility, restoring from backup, setting up database replication until the databases were in synch, taking everything offline for 2-3 hours to merge two servers into one, test everything, and flip the switch.

If only it were that simple.

Thursday Morning, 10 am: An Untimely Crash
We had planned to take the old servers offline at 10 pm Pacific on Thursday, but around 10 am, one of them went down. The server itself was working, but the hard drive was failing, and it wouldn't serve web requests.

I received a few emails from customers asking why the server was down and I was able to explain that this should be the last down-

time for many months. Everyone was appreciative and supportive. But I spent most of the day trying to get the server to stay alive for 12 more hours, with no luck. Half of the users had no access to it for the final 12 hours on the old servers.

Thursday Afternoon, 2 pm: Where's the Data?
We had meticulously planned to have all the data replicated between the old and new databases to ensure they would be in synch when we went to perform the migration. But there was one problem…

Replication had silently failed about 48 hours beforehand, and neither I or my DBA noticed. So at 2 pm, we realized we were literally gigs behind with the synchronization. With only 8 hours until the migration window began, we zipped up these gigs and started copying them from server to server. The dialog said it would take 5.5 hours – no problem! We had 2.5 hours of leeway.

Thursday Night, 8 pm: Copy and Paste
My DBA checked on the copying a few times in the middle of his night (he's in the UK), and it eventually failed with 10% remaining. At that point, we knew that even if we got everything done according to plan, we wouldn't have data for the most recent 48 hours. Grrr.

In a last-ditch effort to get the data across the wire, we selected the info from a single table that changed the most and copied it to the new server, which took only around 5 minutes. The problem was that this data should have been processed by our fancy keyword suggestion algorithm and it hadn't been.

Friday Morning, 1 am: Panic Sets In
It was about this time I began to panic. Trying to process the data using existing code was not going to happen – the algorithm is written in a mix of JavaScript and Classic ASP, which couldn't be executed in the same fashion that it is when the app runs under normal conditions. And it wouldn't run fast enough to process

the millions of rows that needed it in any realistic amount of time.

So I did one of the craziest things I've done in a while. I spent 4 hours, from approximately 11 pm until 3 am, writing a Windows Forms app that combined the JavaScript and Classic ASP into a single assembly – with the JavaScript being compiled directly into a .NET DLL that I referenced from C# code. Then I translated the Classic ASP (VBScript) into C# and prayed it would work.

And after 2 hours of coding and 2 hours of execution, it worked.

Friday Morning, 3 am: It Gets Worse…
During this time, the DBA was trying to merge two databases – copying 75 million rows of data from one DB to another. This was supposed to take 3-5 hours based on tests the DBA had run.

But it was taking 20x longer. The disks were insanely slow because we were running during their backup window, and some of the data was on a shared SAN. By 3 am, when we should have been wrapping up the entire process, we were 10% done with the 75 million rows.

By 6 am, we had to make a call. It was already 9 am on the East Coast, and customers would surely be logging in soon if they hadn't already. I was exhausted and at my wits end with the number of unexpected failures, and I asked my DBA what our options were.

After some discussion, we decided to re-enable the application so users could access it, but continue copying the 75 million rows, but with a forced sleep so the app could run with surprisingly little performance impact. It took four days for all of the data to copy, but we didn't receive a single complaint in the meantime. Lucky for us, no one noticed. And the app running on the new hardware was 2-3x times more responsive than on the old setup.

Friday Morning, 6:45 am: Victory

After some final testing I logged off at 6:45 am Friday morning with a huge sigh of relief. I dashed off the email you read in the intro, then headed to bed to be awoken by my kids an hour later. Needless to say, had a big glass of wine the following night.

Building a startup isn't all the fun you read about on Hacker News and TechCrunch…sometimes it's even better.

THE BIGGEST GAMBLE OF YOUR CAREER[42]

I'm in the midst of the biggest gamble of my career. The thing is, every chance you take feels like your biggest gamble *while you're taking it.*

When I left salaried employment for consulting in 2006, it felt like a big gamble.

When I left consulting for product in 2008 with a wife, kid, and a mortgage, it felt like a huge gamble.

When I spent most of the money I had in the bank on a broken-down SaaS app few people had heard of, called HitTail, I was downright scared. It's hard to put years of work and savings on the line, based on nothing more than confidence that you will execute.

But every one of the above gambles worked out, and I wound up far better off than if I'd never taken them.

And here's the thing…

---

42 Original location: https://robwalling.com/2014/07/31/the-biggest-gamble-of-your-career/

The more gambles you take, the better you become at reading the tea leaves and figuring out if your next gamble is going to work.

You also gain confidence in your ability to execute when your back is to the wall. This means you don't crumble under the stress of the decision.

Instead, you start to become addicted to drawing yourself out of your comfort zone. This is a good thing.

You start to embrace the immense amounts of learning that happen when you're doing something that terrifies you.

Pushing on Drip

When we launched Drip the first month was solid – over $7k in recurring revenue. I was pleased with that result and was looking forward to growth in the months that followed.

Except growth didn't come. Second month revenue (granted, it was December) was $7k.

Third month revenue was the same. And worse yet, I started to see a trend…

Many of the new subscribers I was adding through my tried-and-true marketing efforts were churning out after a few months of use. It's an expression I used to call "leaking out of the bottom of the funnel."

Now we just call it churn.

A high churn rate means you haven't found a market that really needs your product, or, worse yet, that you've built something no market wants (yet).

And if you're paying attention this is the learning phase that happens between building and scaling (I talked about it in my 2013

MicroConf talk). The learning phase is, in my opinion, the most painful part.

So while the percentage of new users who completed onboarding was high for an app of this complexity (close to 60%), they were not sticking around like I would hope, which meant we hadn't hit the nail on the head and solved the problem I came to solve.

Namely: fixing email marketing for startups.

So it was back to work.

We had more than a dozen requests for "marketing automation" features – basically, features that allow you to send subscribers personalized emails based on their behavior.

So if they read a lot of SEO articles on your website, you can tag them with "SEO" and move them into your SEO crash course.

And that's the tip of the iceberg; you can tag people based on links they click, URLs they visit, or things they buy.

Tracking customers from website visitors to marketing subscribers to trial users to customers, and being able to tailor communications the entire time is insanely powerful. In fact, it's the future of email marketing. But I didn't want to build a behavioral email/marketing automation platform.

It would be a *ton of work* to build, and the space has been crowded for years. It's not a typical bootstrapper play, it's the kind for which you raise funding.

But the more founders I spoke with, the more I realized this was the real problem they were facing. Most founders have either outgrown their first email marketing app (think of a static email newsletter service like MailChimp or AWeber), or they've tried a beefy tool like Infusionsoft and it hasn't worked out due to high

price and low usability.

Contrary to my plans, founders didn't need a *little* help with email marketing, they were telling me they needed a massive step up from the tools on the market.

Perhaps fixing email marketing for startups was a much bigger problem than I had originally thought. And perhaps this new approach was the one that I came here to solve.

Best Laid Plans
We planned for 1 month of development to build email marketing automation focused on the Saas, WP plugin, and info product use cases. It took almost 5.

We rolled it out in batches – a new feature every few weeks. And the strangest thing happened as we did…

Revenue started moving up. First, a few hundred. Then a thousand.

Churn plummeted. By more than 50%. Revenue rose almost 50% in a couple of months.

All during a time when I was heads down, not doing a lick of marketing.

I had a feeling we were on to something.

The Gamble
I had invested a year's worth of company profits into Drip. Funds that could buy a house in most cities in the world. And I had invested the bulk of my waking hours for a year.

We were entering a competitive market with massive upside and players who had raised millions of dollars.

And there I sat, on the eve of unleashing *Drip 2.0*, realizing that

this was, once again, the biggest gamble of my career.

The thing is, every chance you take feels like your biggest gamble *while you're taking it.* And most of the time, choosing the gamble is the right choice.

WHAT I LEARNED BUYING, GROWING, AND SELLING MY STARTUP[43]

In early 2011 I was looking for my next thing. Long ago, I learned that when I'm not learning I'm not happy. And in early 2011, aside from hosting our first successful MicroConf, I wasn't doing many things that scared me.

Which told me I needed a *next thing*.

My book was selling well. I had a portfolio of 8 or 9 small apps and websites. But nothing was pushing me to expand beyond my current mental limits, and I knew that within months I would start to feel a tinge of burnout and apathy that would eventually turn into genuine unhappiness.

I had spent the previous 10 months hanging out with our newborn, working 12-16 hours a week, and making an income comparable to what I had made working full-time.

Life was good, and I had to go screw it up.

The Acquisition

With patience as one of my least prominent traits, I didn't want

43 Original location: https://robwalling.com/2015/12/02/what-i-learned-buying-growing-and-selling-my-startup/

to spend 6 months coding and 12 months launching/learning/trying to find a product/market fit.

So I took a page out of my playbook and went on a search for an acquisition larger than any I had previously attempted.

I've already detailed the acquisition in a 3-part blog series, so I won't rehash it here. But suffice to say, I spent almost every penny in my business account to buy a SaaS app that looked like this:

And was able to turn it into this:

And 10x the revenue in 15 months, via steps I outlined in a Micro-Conf talk a few years ago.

Things were good while I was growing HitTail. I was often frustrated with how long each step took, but I was learning huge lessons along the way.

Another Next Thing

And then things changed…and I had another "what's next?" moment.

This was partially due to Google pulling *Not Provided* out of its hat in 2012/2013 (which we worked around pretty easily, but it did scare me for a couple of months). And partly due to the massive amount of learning, I had packed into the revamp and growth of HitTail.

It became apparent during one of my semi-annual retreats that I needed to find my next thing. Again.

The next thing turned out to be Drip. I won't go into its origins here since I've detailed them elsewhere. But suffice to say, at a certain point, Drip's growth curve became such that it didn't make sense for me to focus even a few hours a month on any other products.

So I began divesting myself of my portfolio of websites and web applications.

I shut a few of them down. I sold one for a few thousand dollars when someone made me an offer out of the blue. I gave my share of another app to a business partner.

And then there was HitTail. It was making too much money to shut down or give away. But luckily, this time, a secondary market for web applications had started to develop.

Finally, A Secondary Market for SaaS Apps

A few years ago, the options for selling a small web application were bleak. I didn't know of any reputable brokers. Flippa is akin to selling wholesale (and dealing with a lot of uneducated buyers in the process).

But over the past 4 years, I've watched first-hand as a secondary market for small websites and web apps has developed through reputable brokers like FE International and Quiet Light.

Having developed a relationship with FE International over the past several years, I chose them to assist with the HitTail sale.

And a few months later (after what felt like hundreds of hours of due diligence work, but was more like 40-50), I found myself staring at a newly augmented bank balance and my list of applications that had once approached a dozen in number….now standing at 1.

It's all about Drip, baby. And that focus feels great.

So What Did I Learn?

As soon as news of the sale hit the front page of Hacker News, questions started coming in: Would you do it again? Was it a financial win? What did you learn?

Here are a few things:

Lesson #1: "Stair-Stepping" Is Very Much Alive
I've come up with a name for this approach of starting small and working your way up through larger and larger opportunities as you gain experience, skills, funds, and confidence. I call it Stair Stepping.

This move from HitTail to Drip marks my next stair-step, from step 3 (recurring revenue) into an as-yet-undefined step 4 (might be: more recurring revenue in a larger and more competitive market).

Had I tried to launch Drip in 2011 without the experience, skills, funds, and confidence I gained from growing HitTail, I would have had my ass handed to me.

Launching and growing Drip has been hard. I honestly don't think I could have pulled it off P.H. (pre-HitTail).

The more I learn, the more I see Stair Stepping as an optimal approach for maximizing results while minimizing your chance of failure. It's not the fastest path, but a predictable one.

Lesson #2: Money Makes Things Easier
I was doing fine before HitTail. As I said, I was working 12-16 hour weeks, hanging out with my kids during my non-work hours. And making a full-time income (akin to what I made as a contract developer in California).

But one night, in the middle of a conversation about earnings, money, and other financial stuff, I asked my wife: "What would having more money allow us to do?"

She rattled off a number of things, but the ones that really hit me were:

- Be more generous

- Travel
- Save for the kids' college
- Have more control of our schedules
- Feel more security about the financial present/future

Notice none of the things she mentioned involved *buying stuff*. Aside from nerdy t-shirts and good whiskey, I don't care much for *stuff.*

So I hadn't given much thought to making more money once I'd hit that "good enough" mark back in 2008/2009. But this conversation changed the game for me.

Within a year, HitTail had changed our standard of living and accomplished most of the things she mentioned during that conversation, including allowing us to help friends and acquaintances out of a few financial situations, bringing the kids with us for a month in Europe/Thailand each year for the past 3 years, and generally feeling more definite about our financial future.

I'm not in a place where it would be possible for me to never work again. But HitTail was a noticeable step towards that possibility. It gave me a glimpse of what that might feel like.

Let me be clear: I don't think you need buckets of money to be happy. And, in my experience, "more stuff" is a ridiculous reason to seek more money.

But you can do some crazy generous, and interesting things when you have a bit more breathing room.

Lesson #3: Buying is Probably Better Than Building
I've been singing this tune for a long time. But acquiring an application that already has product/market fit puts you 12-18 months

ahead of starting from scratch.

Like most founders, I'm impatient. And leaping ahead is so much easier than building something from nothing.

By my estimation, I think acquiring HitTail condensed my career timetable by 18 months. And removed a lot of anxiety and stress of the pre-product-market fit thrashing.

#4: A Team of One Can Accomplish Crazy Things
The amount of leverage one can achieve given marketplaces like Upwork, metered hosting like Amazon's AWS, inexpensive payment gateways like Stripe, and the myriad of other tools that are multiple orders of magnitude cheaper than they were a decade ago, is mind-blowing.

My journey with HitTail would have been impossible 10 years before.

I believe today is the best time in history to be a founder. And it's especially easier for a single founder given the power of outsourcing and today's global labor pool.

Epilogue
The day the HitTail sale closed, I received a few congratulatory texts from close friends.

At this point, after weeks of due diligence, I was emotionally exhausted. Selling something of this scale is hard. You don't think it's going to be that hard, but it is.

There's an emotional component as you constantly wonder if the sale is going to fall through. You don't sleep well. And you have a hard time letting go of something into which you've invested so much time, effort, and emotion.

But sometimes, it's the right thing to do.

EXTRAS

THE 3 ASPECTS OF A GREAT CONFERENCE TALK[44]

When I started public speaking in 2008 I was really bad. Nervous. Content didn't land. Often times I didn't "get" my audience.

I spoke a few times in 2008/2009, but it wasn't until 2010 that I made strides towards getting better. As I did it I stumbled upon a framework of how I view public speaking.

My mental model of a talk has 3 aspects that should work in tandem:

- Content
- Delivery
- Entertainment

Content
Content can be inspirational, aspirational, tactical…maybe a few others?

The mistake I made early on was making my content too tacti-

44 Original location: https://robwalling.com/2019/09/30/the-3-aspects-of-a-great-conference-talk/

cal and ignoring delivery and entertainment. So my talks came across as very dry.

This was a reaction to watching speakers like Tony Robbins give talks almost completely devoid of actionable content, but his delivery and entertainment levels were off the charts.

And that bothered me. A lot. So I over-corrected in the other direction.

The key with content is to know your audience and their expectations.

If Tony Robbins took the stage at one of his events and gave an amazing in-depth tutorial on how to run Facebook ads, people would mutiny.

Because that's not what they expect from Tony Robbins.

I think Tony is a phenomenal speaker for the RIGHT audience (those who need motivation, not those who need specific tactics). Gary V. is similar.

Delivery
Delivery is about stage presence. Energy. Confidence.

You know it when you see it – some people have great stage presence, while other people (like me in 2008) seem nervous and meek. Over-caffeinated and stressed.

This one takes a lot of repetition to overcome (for those who don't have it naturally), whether on your own or with a speaking coach.

Entertainment
Entertainment is the third piece.

If you are funny on stage…I hate you.

I have never been able to pull off "funny." I am jealous of speakers like Patio11 and Lianna Patch, who can rattle off joke after joke from the stage. This isn't me.

If it's not you, my best advice is to lean into being yourself. For me, this turned into telling super nerdy jokes about programming and physics.

Or showing 60-second videos during a short intermission in my talk. I've struggled to be funny on stage so I've taken a different approach that's more in line with who I am.

Not saying it works, but it's the best I've been able to pull off.

After running MicroConf for nearly 10 years we've had more than 200 speakers on stage (including attendee speakers).

With MicroConf, I've noticed the best talks have amazingly applicable content, good delivery (doesn't have to be amazing), and a bit (but not too much) entertainment.

But your mileage may vary based on the audience.

Out of Balance

What's interesting is if you imagine a talk with only great content (but poor delivery and no entertainment) – it's super dry.

Great delivery and entertainment but shallow content? That's a motivational speech.

Great entertainment but vapid content and poor delivery? That might feel like a poor stand-up comedy routine.

I have a hard time imagining a great talk that doesn't have a dose of each of these 3 aspects.

OTHER WORKS BY THE AUTHOR

Start Small, Stay Small

A Developer's Guide to Launching a Startup

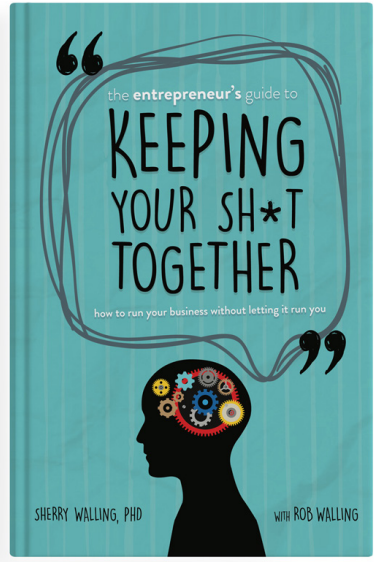
Start Small, Stay Small focuses on practical, step-by-step instructions used by hundreds of developers on the road to launching their startups. Whether you have a product idea or are still looking, this book takes you through the process of finding an idea, testing it, converting visitors to buyers, and attaining profitability as quickly as possible.

"Every software engineer can pull an idea out from here…
I wish I had this when I was starting my business."

— PATRICK MCKENZIE, KALZUMEUS

The Entrepreneur's Guide to Keeping Your Sh*t Together

How to Run Your Business Without Letting it Run You



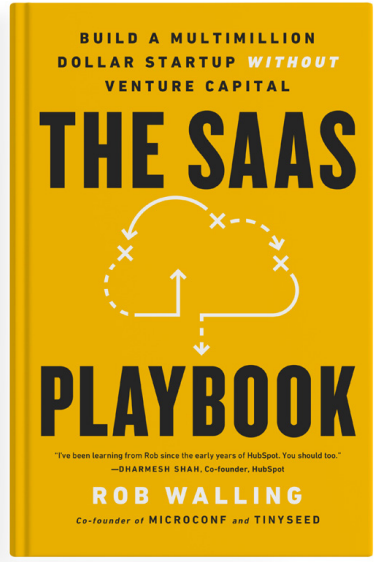
Running a business is hard. It can ruin your health, your relationships, and your life. *The Entrepreneur's Guide to Keeping Your Sh*t Together* is an invaluable guide to staying sane and ensuring both you and your business thrive for years. Learn new ways to deal with the responsibility and fear of being an entrepreneur; cope with depression, anxiety, burnout, ADHD, and other common psychological burdens; and overcome procrastination to get more things done—faster.

"A personal, generous, and incredibly useful guide to staying sane and changing the world at the same time. Read it before you think you need it."

— SETH GODIN

The SaaS Playbook

Build a multimillon dollar startup without venture capital



Do you want to build a software product that people want and are willing to pay for? Do you dream of growing the kind of company that employs smart, creative people?
This book will walk you through the most actionable tactics to bootstrap a multimillion-dollar SaaS business. Learn how to compete against large competitors and structure your pricing for maximum growth. Discover the four SaaS Cheat Codes that can dramatically accelerate your startup's success. Learn how to avoid common mistakes that SaaS founders make and identify the right marketing approaches for your business.

"Rob knows software, and he knows startups. I've been learning from him since the early years of HubSpot. You should too."

— DHARMESH SHAH
Co-founder/CTO, HubSpot