Startup Lessons Learned

All Seasons

"the lean startup."

Eric Ries

Startup Lessons Learned

All Seasons: Every Post from the Startup Lessons Learned Blog

This version was published on 2012-05-24

Tweet This Book!

Please help Eric Ries by spreading the word about this book on Twitter!

The suggested hashtag for this book is #startuplessonslearned.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#startuplessonslearned

Contents

August 2008 **1**

Paul Graham on fundraising 1

Refactoring for TDD and interaction design 3

September 2008 **5**

Not crossing the chasm 5

Customer Development Engineering 6

Smarticus — 10 things you could be doing to your code right now 9

October 2008 **11**

What does a startup CTO actually do? 11

About the author 16

The product manager's lament 19

When NOT to listen to your users; when NOT to rely on split-tests 23

The App Store after the gold rush 29

Three decisions to make on virtual goods 32

The engineering manager's lament 38

Lean startups vs lean companies 46

Chuck's Code & Learning: Learning to write tests that matter 49

A hierarchy of pitches 49
John Doerr’s 10 lean startup tips 53

July 2010 **56**
The Entrepreneur’s Guide to Customer Development 56
Founder personalities and the “first-class man” theory of management 60
Some IPO speculation 64
Case Study: kaChing, Anatomy of a Pivot 68
Case Study: SlideShare goes freemium 74

August 2008

Paul Graham on fundraising

I have found no better primer on the current realities of starting a new technology company in a startup hub like Silicon Valley than Paul Graham's essays. Alas, they aren't published in a dead-tree medium yet, so I can't say something like "they are the essential reference on my bookcase..." but rest assured they would be.

A Fundraising Survival Guide[1]

> Raising money has a mysterious capacity to suck up all your attention. Even if you only have one meeting a day with investors, somehow that one meeting will burn up your whole day. It costs not just the time of the actual meeting, but the time getting there and back, and the time preparing for it beforehand and thinking about it afterward.
>
> What I tell most startups we fund is that if someone reputable offers you funding on reasonable terms, take it. There have been startups that ignored this advice and got away with it—startups that ignored a good offer in the hope of getting a better one, and actually did. But in the same position I'd give the same advice again. Who knows how many bullets were in the gun they were playing Russian roulette with?

[1]http://www.paulgraham.com/fundraising.html

The Hacker's Guide to Investors[2]

> Whatever help investors give a startup tends to be underestimated. It's to everyone's advantage to let the world think the founders thought of everything. The goal of the investors is for the company to become valuable, and the company seems more valuable if it seems like all the good ideas came from within.
>
> I say this as a founder: the contribution of founders is always overestimated. The danger here is that new founders, looking at existing founders, will think that they're supermen that one couldn't possibly equal oneself. Actually they have a hundred different types of support people just offscreen making the whole show possible.

How to Fund a Startup[3]

> It's obvious why investors delay. Investing in startups is risky! When a company is only two months old, every *day* you wait gives you 1.7% more data about their trajectory. But the investor is already being compensated for that risk in the low price of the stock, so it is unfair to delay.
>
> Fair or not, investors do it if you let them. Even VCs do it. And funding delays are a big distraction for founders, who ought to be working on their

[2]http://www.paulgraham.com/guidetoinvestors.html
[3]http://www.paulgraham.com/startupfunding.html

> company, not worrying about investors. What's a startup to do? With both investors and acquirers, the only leverage you have is competition. If an investor knows you have other investors lined up, he'll be a lot more eager to close– and not just because he'll worry about losing the deal, but because if other investors are interested, you must be worth investing in. It's the same with acquisitions. No one wants to buy you till someone else wants to buy you, and then everyone wants to buy you.
>
> The key to closing deals is never to stop pursuing alternatives. When an investor says he wants to invest in you, or an acquirer says they want to buy you, *don't believe it till you get the check.* Your natural tendency when an investor says yes will be to relax and go back to writing code. Alas, you can't; you have to keep looking for more investors, if only to get this one to act.

Refactoring for TDD and interaction design

In TDD, we follow a rhythm of "test-code-refactor." This basic pattern is useful in all aspects of product development. The basic idea is to avoid building something based on what you think it might need to do in the future. Instead, we build for today, but then constantly look for ways to reconfigure what we've created to make it more general, more flexible, more useful. This process is called refactoring. In code, it helps us to build reusable components that are suitable for assembly into complex systems without having to guess which components are needed (or how they should work). In effect, the specific examples drive the spec for what the general pieces should look like.

The same process works in Interaction Design. The temptation is try and build a toolkit of general pieces, which can be assembled into many kinds of final product. This rarely works, because it is hard to build pieces that work optimally for diverse uses without damaging each use case (which leads to mediocre or compromised results). Much better is to infer the right toolkit by searching for commonalities among actually-built experiences.

September 2008

Not crossing the chasm

What does life feel like in the chasm[4]? How do you plan for it? A growing startup with a well-run product team will have a history of steady progress. Incremental feature releases leading to correlated growth. The strength of the team determined the pace, and the best companies iterate and learn fastest.

Now imagine it just stops working. You execute just like before. You delight customers just like before. You listen and learn. You innovate as well as ever. Nothing seems to matter. In a subscription business, maybe your attrition starts matching your acquisition, balancing like magic. In an eyeballs business, you just can't seem to acquire or activate that next step-up of customers. Or your cost of customer acquisition just magically floats up to match your customer lifetime value.

We talk a good game about technology adoption curves and crossing the chasm, but most of us don't seem to recognize when it's happening. we don't talk enough about how it feels. Things just stop working. It's everybody's fault and nobody's too. The result: frustration, impotence, humiliation.

My two cents: talk about this beforehand. It's going to be hard to talk about in the midst of the malaise. Be ready to drive home this simple point: in a phase change the things that made you successful before don't work anymore.

[4]http://en.wikipedia.org/wiki/Crossing_the_Chasm

Customer Development Engineering

Yesterday, I had the opportunity to guest lecture again in Steve Blank[5]'s entrepreneurship class at the Berkeley-Columbia executive MBA program[6]. In addition to presenting the IMVU case, we tried for the first time to do an overview of a software engineering methodology that integrates practices from agile software development[7] with Steve's method of Customer Development[8].

I've attempted to embed the relevant slides below. The basic idea is to extend agile, which excels in situations where the problem is known but the solution is unknown, into areas of even greater uncertainty, such as your typical startup. In a startup, both the problem and solution are unknown, and the key to success is building an integrated team that includes product development in the feedback loop with customers.

As always, we had a great discussion with the students, which is helping refine how we talk about this. As usual, I'm heavy on the theory and not on the specifics, so I thought I'd share some additional thoughts that came up in the course of the classroom discussion.

1. Can this methodology be used for startups that are not exclusively about software? We talk about taking advantages of the incredible agility offered by modern web architecture for extremely rapid deployment, etc. What

[5] http://www.haas.berkeley.edu/faculty/blank.html

[6] http://www.berkeley.columbia.edu/

[7] http://agilemanifesto.org/

[8] http://www.amazon.com/gp/product/0976470705?ie=UTF8&tag=lessolearn01-20&linkCode=as2&camp=1789&creative=9325&creativeASIN=0976470705

about a hardware business with some long-lead-time components?

To be clear, I have never run a business with a hardware component, so I really can't say for sure. But I am confident that many of these ideas still apply. One major theory that has influenced the way I think about processes comes from Lean Manufacturing[9], where they use these same techniques to build cars. If you can build cars with it, I'm pretty sure you can use it to add agility and flexibility to any product development process.

1. What's an example of a situation where "a line of working code" is not a valid unit of progress?

This is incredibly common in startups, because you often build features that nobody wants. We had lots of these examples at IMVU, my favorite is the literally thousands of lines of code we wrote for IM interoperability. This code worked pretty well, was under extensive test coverage, worked as specified, and was generally a masterpiece of amazing programming (if I do say so myself). Unfortunately, positioning our product as an "IM add-on" was a complete mistake. Customers found it confusing and it turned out to be at odds with our fundamental value proposition (which really requires an independant IM network). So we had to completely throw that code away, including all of its beatiful tests and specs. Talk about waste.

1. There were a lot of questions about outsourcing/offshoring and startups. It seems many startups these days are

[9] http://www.amazon.com/gp/product/0743249275?ie=UTF8&tag=lessolearn01-20&linkCode=as2&camp=1789&creative=9325&creativeASIN=0743249275

under a lot of pressure to outsource their development organization to save costs. I haven't had to work this model under those conditions, so I can't say anything definitive. I do have faith that, whatever situation you find yourself in, you can always find ways to increase the speed of iteration. I don't see any reason why having the team offshore is any more of a liability in this area than, say, having to do this work while selling through a channel (and hence, not having direct access to customers). Still, I'm interested in exploring this - some of the companies I work with as an advisor are tackling this problem as we speak.

2. Another question that always comes up when talking about customer development, is whether VCs and other financial backers are embracing this way of building companies. Of course, my own personal experience has been pretty positive, so I think the answer is yes. Still, I thought I'd share this email that happened to arrive during class. Names have, of course, been changed to protect the innocent:

   > Hope you're well; I thought I'd relay a recent experience and thank you.
   >
   > I've been talking to the folks at [a very good VC firm] about helping them with a new venture ... Anyway, a partner was probing me about what I knew about low-burn marketing tactics, and I mentioned a book I read called "Four Steps to the..."
   >
   > It made me a HUGE hit, with the partner explaining that they "don't ramp companies like they used to,

> and have very little interest in marketing folks that don't know how to build companies in this new way."

Anyway, thanks to Steve and all of his students - it was a fun and thought-provoking experience..

Smarticus — 10 things you could be doing to your code right now

Smarticus — 10 things you could be doing to your code right now[10]

A great checklist of techniques and tools for making your development more agile, written from a Rail perspective. Of the techniques he mentioned, I think four are fundamental and critical for any lean startup:

TDD (or the even more politely named TATFT[11])
Continuous integration
Automate your deployments
Collect statistics

The tools to help you do these things are getting better and better every day, but don't confuse tools with process. Whatever state your code or team is in, you can always start going faster. Or

[10]http://smartic.us/2008/9/9/10-things-you-could-be-doing-to-your-code-right-now

[11]http://smartic.us/2008/8/15/tatft-i-feel-a-revolution-coming-on

to borrow from a military context (John Boyd[12]) “people-ideas-hardware, in that order[13].”

[12]http://www.amazon.com/gp/product/0316796883?ie=UTF8&tag=lessolearn01-20&linkCode=as2&camp=1789&creative=9325&creativeASIN=0316796883

[13]http://www.d-n-i.net/fcs/comments/c418.htm

October 2008

What does a startup CTO actually do?

What does your Chief Technology Officer do all day? Often times, it seems like people are thinking it's synonymous with "that guy who gets paid to sit in the corner and think 'technical' deep thoughts" or "that guy who gets to swoop in a rearrange my project at the last minute on a whim." I've tried hard not to live up (or down?) to those stereotypes, but it's not easy. We lack a consistent and clear definition of the job.

When I've asked mentors of mine who have worked in big companies about the role of the CTO, they usually talk about the importance of being the external face of the company's technology platform; an evangelist to developers, customers, and employees. That's an important job, for sure, and I've been called upon to do it from time to time. But I don't think most startups really have a need for someone to do that on a full time basis.

So what does CTO mean, besides just "technical founder who really can't manage anyone?"

I always assumed I wouldn't manage anybody. Being a manager didn't sound fun - deep down, who really wants to be held accountable for other people's actions? I mean, have you seen other people? They might do anything! So I initially gravitated to the CTO title, and not VP of Engineering. I figured we'd bring in a professional to do the managing and scheduling-type stuff, and I could stay focused on making sure we built really awesome technology. But along the way, something strange happened. It became harder and harder to separate how the software is built from how the software is structured. If you're trying to design an architecture to maximize agility, how can that work if some people are working in TDD and others not? How can it work if some folks are pre-building and others use five why's to drive decisions? And what about if deployment takes forever? Some options can improve the performance of the softare at the expense of readability, deployability, or scalability. Should you take them? These sounded to me like technical problems, but when you do any kind of root cause analysis they turn out to be people problems. And there's really no way to tackle people problems from the sidelines.

So I wound up learning the discipline of managing other people. Turns out, I wasn't too bad at it, and I found out just how rewarding it can be. But since I spent a long time in a hybrid CTO/VP Engineering role, I still have this nagging question. Just what is the CTO supposed to do?

Here's my take. The CTO's primary job is to make sure the company's technology strategy serves its business strategy. If that sounds either too simple or too generic, think for a second if any companies you know do the reverse. Have you ever heard a technologist use technical mumbo-jumbo to make it sound like a business idea he or she didn't like was basically impossible? That's what we should be trying to avoid.

I'll try and break it down into five specific skills.

- Platform selection and technical design - if your business strategy is to create a low-burn, highly iterative lean startup, you'd better be using foundational tools that make that easy rather than hard. Massive proprietary databases? I don't think so. Can the company dig into its tools when they fail and fix them? If not, who's going to insist we switch to free and open source software? When projects are getting off the ground, who can the team check with to make sure their plans are viable? Who will hold them accountable for their project's impact on the platform as a whole?

- Seeing the big picture (in graphic detail) - the CTO should be the person in the room who can keep everything your technology can and can't do in their head. That means knowing what's written and what's not, what the architecture can and can't support, and how long it would take to build something new. That's more than just drawing architecture diagrams, though. Being able to see the macro and micro simultaneously is a hallmark of all of the really great technologists I've had the privilege to work with.

- Provide options - another mark of a good CTO is that they never say "that's impossible" or "we'd never do that." Instead, they find options and can communicate them to everyone in the company. If the CEO wants to completely change the product in order to serve a new customer segment, you need someone in the room who can digest the needs of the new (proposed) business, and lay out the costs of each possible approach. Some technologists have a tendency just to "decide for you" and give you the "best" option, but that's dangerous. You can't have an honest dialog if one party knows all the answers.

- Find the 80/20 - this was my favorite part of the job. Sometimes, you're in a meeting where someone wants to build a new feature. And in their mind, they've got it all spec'ed out. It slices, it dices, and probably washes your car too. In my mind, they're racking up costs (one month for that part, two months for that other part, uh oh). On a bad day, I'd just give them the sobering news. But a good day looked like this. Once I understood what the objective of their feature was for customers, I could sometimes see a way to get 80% of the benefit for 20% of the cost. "Would you be able to learn what you need to learn if that feature just sliced, but not diced? Because if we don't have to add a dicing module, we can repurpose the flux capacitor via solar flares...." I was constantly amazed how often the answer was something like "really? dicing is what's expensive?! I just threw that in there on a whim!"

- Grow technical leaders - I like to formalize this responsibility by eventually designating some engineers as "Technical Leads" and delegating to them the work of guiding the technical direction of more and more projects. This is the only way to scale. It also forced me to get clear about which aspects of our company's technical direction were really important principles, and which were just artifacts of how we got there. With multiple people trying to work to the same standard, we had to be a lot crisper in our definitions. Was the fact that we were primarily using PHP essential, or could we add new tools written in other languages? Was it an important or irrelevant fact that most of our web code was procedural and not object-oriented? What if someone wanted to write their module in OOP style? By delegating and training, we create a corps of leaders who could step in to provide CTO-like services on demand. And by working together, we created a team whose whole was greater than the sum of its parts[14].

I want to add one last idea, even though I recognize it is controversial, bordering on the boundary between the CTO and VP Engineering. I don't know how much I'm being influenced by having worn both hats, but I think it's important enough to go out on a limb and add.

- Own the development methodology - in a traditional product development setup, the VP Engineering or some other full-time manager would be responsible for making sure the engineers wrote adequate specs, interfaced well with QA, and also run the scheduling "trains" for releases. But I think in a lean startup, the development methodology is too important to be considered "just management." If the team is going to use TDD or JIT scalability[15], for example, these choices have enormous impact on what the architecture must look like. At a minimum, I think it's the CTO and Tech Leads that have to be responsible for five why's-style root cause analysis of defects. Otherwise, how can they find out what their blind spots are and make sure the team and the architecture is adjusting? That job calls for someone who sees the big picture.

Your CTO might be a great architect, evangelist, interface designer or incredible debugger. Those are great skills to have, and I'm curious what you've seen work and not work. I'll be the

[14]http://vanillabomb.files.wordpress.com/2008/04/voltron.jpg

[15]http://startuplessonslearned.blogspot.com/2008/09/just-in-time-scalability.html

first to admit that my experience is limited, so I'm collecting anecdotes. Have you worked with or for a great CTO? What made them exceptional? What's one thing a brand-new first-time CTO could learn from them?

About the author

The most common feedback I've heard from readers[16] has been that I should provide details on my background. I didn't include much on the blog at first, because I want you to judge what I write based on what I say, rather than who I am. So if you're new, consider not paying any attention to the rest of this post, and just diving into the archives, if you haven't already. (Maybe you'd like to start with The lean startup[17], How to listen to customers[18], or What does a startup CTO actually do?[19])

For everyone else, here's the standard bio paragraph I use for conferences and other formal occasions:

> Eric Ries[20] became a Venture Advisor at Kleiner Perkins Caufield & Byers, after co-founding and serving as Chief Technology Officer of IMVU. He is the co-author of several books including The Black

[16]http://startuplessonslearned.blogspot.com/2008/09/lo-my-5-subscribers-who-are-you.html

[17]http://startuplessonslearned.blogspot.com/2008/09/lean-startup.html

[18]http://startuplessonslearned.blogspot.com/2008/09/how-to-listen-to-customers-and-not-just.html

[19]http://startuplessonslearned.blogspot.com/2008/09/what-does-startup-cto-actually-do.html

[20]http://startuplessonslearned.blogspot.com/

> Art of Java Game Programming[21] (Waite Group Press, 1996). While an undergraduate at Yale Unviersity, he co-founded Catalyst Recruiting. Although Catalyst folded with the dot-com crash, Ries continued his entrepreneurial career as a Senior Software Engineer at There.com, leading efforts in agile software development and user-generated content. In 2007, BusinessWeek named Ries one of the Best Young Entrepreneurs of Tech[22]. He serves on the advisory board of a number of technology startups including pbWiki, Bunchball, FooMojo, Causes and KaChing.

I'm one of those people who's been programming since they can remember. I got my start programming on an old IBM XT; it was thanks to MUDs[23] that I first discovered the internet. Those early text-based games were programmed by their own users, and it was by far the best tutorial I could ever have received in the power of software. In a MUD, you could literally conjure new objects that never existed before, just by programming them. I know many people who think that software works like magic, but to me it actually was magic.

Later, I discovered you could get paid to program computers, and really never looked back. While I was still in high school, I became a Java "expert" during a time when there was no such thing. Thanks to Sun's amazing PR blitz, there was tremendous

[21] http://www.amazon.com/gp/product/1571690433?ie=UTF8&tag=lessolearn01-20&linkCode=as2&camp=1789&creative=9325&creativeASIN=1571690433

[22] http://www.businessweek.com/technology/special_reports/20070326techsbesty.htm

[23] http://en.wikipedia.org/wiki/MUD

demand for experts on Java, and I did my best to convince people that I was one of that mythical breed. Thanks to the anonymity of the internet[24], I landed a few jobs, and did quite a bit of writing.

By the time the entrepreneurial bug hit me, the dot-com boom was in its waning days. So much for timing. But I managed a few "good learning experiences" before throwing myself full-bore into IMVU. For almost five years I had the opportunity to build and serve with one of the most talented team I have ever seen. It was by far the most intense and most rewarding experience of my professional life. Because of IMVU's reputation, I've also had the opportunity to serve as an advisor or board member for more than a dozen startups. Rolling up my sleeves and serving with them has enriched my understanding and provided many of the lessons I write about here.

In retrospect, there are some clear themes that stand out from across my career. I have always tried to be a consistent advocate for rapid iterations, fact-based decision making, free software, and values-centric organizations. Every startup has a chance to change the world, by bringing not just a new product, but an entirely new institution into existence. That institution will touch many people in its life: customers, investors, employees, and everyone they touch as well. I believe we have an obligation to ensure the resulting impact is worthy of the energies we invest in bringing it to life.

Eventually, I came to summarize these themes with the phrase

[24]http://en.wikipedia.org/wiki/On_the_Internet,_nobody_knows_you%27re_a_dog

"the lean startup[25]." Lean[26] is one of the major trends shaping our world, and its impact goes beyond just optimizing our supply chains. Lean startups[27] can be the most capital efficient companies in the world, because they strive to prevent energy from being expended uselessly. Human talent, passion, and wisdom is too precious a commodity to allow it to be wasted.

So that's me, your author. I hope you take something of value from this blog. If you do, please share your story here in a comment.

Thanks for stopping by.

The product manager's lament

Life is not easy when you're working in an old-fashioned waterfall development process, no matter what role you play. But I have a special sympathy for the "product manager" in a startup that is bringing a new product to a new market, and doing their work in large batches. I met one recently that is working on a really innovative product, and the stories I heard from their development team made me want to cringe. The product manager was clearly struggling to get results from the rest of the team. These are smart people trying hard to all row in the same direction. So why are they having so much difficulty?

Let's start with what the product manager does. He's supposed to be the person who specifies what the product will do. He

[25]http://startuplessonslearned.blogspot.com/2008/09/lean-startup-comes-to-stanford.html

[26]http://en.wikipedia.org/wiki/Lean_manufacturing

[27]http://startuplessonslearned.blogspot.com/2008/09/lean-startup.html

writes detailed specs which lay out exactly what features the team should build in its next iteration. These specs are handed to a designer, who builds layouts and mockups of all the salient points. Then the designs are handed to a team of programmers with various specialties. Each specialist takes up his part of the spec (UI, middleware, backend) and cranks out code. Last, the QA team builds a test plan based on the spec, and then tests the features to make sure they conform to the plan.

This system naturally lends itself to a pipeline approach, which the product manager organizes. While the programmers are off building the next major feature, he is busy writing specs so that, when they finish, there won't be any idle time before they can start the next iteration. If the programmers go idle, it's bad news for the product manager, because he's supposed to keep them busy building product. Otherwise, the company is wasting serious money.

When I met this team, some acrimony had built up. The last few features came out pretty different from what was origianlly spec'd, and took far too long, to boot. The programmers keep asking for more say in the designs and direction that they work on. So the team has been spending more and more time in the spec and design phases, trying to get team buy-in on what they are going to build. For some reason, though, despite the increased buy-in, the final product often doesn't look anything like the original spec. The VP Engineering spends all of his time trying to make sure the programmers understand and implement the spec. Each iteration takes longer than the previous one. Frustration is mounting.

It doesn't take long to discover that the product manager is being forced to write every spec five times. First, he writes it nice and clear. Next he works with the designer to build out the design spec, and with QA to build out the test plan. When the programmers get it, they often start negotiating with him about what's going to be built. They exchange countless emails, and he's being constantly interrupted and being asked to clarify exactly what the spec means. The fourth spec exists only in these emails, which are changing the design in an ad-hoc fashion. By the time QA gets the feature, their test plan is badly out of date. So the product manager winds up actually having to use the software, by hand, updating the spec and helping create a new test plan. Naturally, the deviations from the spec are so severe, that he has to rewrite the spec he's currently working on (for the next major feature) to take them into account.

Ironically, this system was designed to keep each functional group 100% utilized, so nobody goes idle, including the product manager. But as the iterations get longer, he's spending more and more of his time answering questions. The interruptions are so bad, he has to write his new specs at 3AM, just to keep the pipeline full.

What's wrong with this picture? Everyone is working at 110%, with full dedication. But the team is falling further and further behind.

Here are the changes I'm working with this team to implement

- Work in cross-functional teams. Each team has a representative of each function. To start, we'll try a product manager, designer, programmer or two, and QA. The team owns a complete feature end-to-end.
- Focus on speed of iteration rather than utilization of every function. Let people go idle, if they can't help the current

iteration succeed. I'm contiuously amazed how many people have untapped creativity and resourcefulness. They don't want to be idle. By letting them focus on the success of their team exclusively, you empower them to do whatever it takes to make the team successful. Will that mean someone in design will jump in to help QA get the release certified? We'll see.

- Switch the spec process from push to pull. Start with a one-page spec, no more. Then, let the team ask questions of the product manager whenever they need clarification. In exchange, the team agrees to show each piece of working code to the product manager for his approval. They'll find points of disagreement much faster, and resolve them in realtime. Plus, the team will get better and better at interpreting the concise specs that only have to be written once. (Eventually, they may abolish specs altogether)

There's much more this team can do to eliminate waste in the way that they work and thereby iterate faster. Eventually, I hope to get them on a full agile diet, with TDD, scrums, sprints, pair programming, and more. But first I think we need to save the product manager from that special form of torture only a waterfall product development team can create. Once the different parts of the team can trust each other again, we'll have the basis we need to start a true continuous improvement feedback loop.

When NOT to listen to your users; when NOT to rely on split-tests

There are three legs to the lean startup[28] concept: agile product development[29], low-cost (fast to market) platforms[30], and rapid-iteration customer development[31]. When I have the opportunity to meet startups, they usually have one of these aspects down, and need help with one or two of the others. The most common need is becoming more customer-centric. They need to incorporate customer feedback[32] into the product development and business planning process. I usually recommend two things: try to get the whole team to start talking to customers ("just go meet a few") and get them to use split-testing[33] in their feature release process ("try it, you'll like it").

However, that can't be the end of the story. If all we do is mechanically embrace these tactics, we can wind up with a disaster. Here are two specific ways it can go horribly wrong. Both are related to a common brain defect we engineers and entrepreneurs seem to be especially prone to. I call it "if some is good, more is better" and it can cause us to swing wildly from one extreme of belief to another.

[28]http://startuplessonslearned.blogspot.com/2008/09/lean-startup.html

[29]http://startuplessonslearned.blogspot.com/2008/09/customer-development-engineering.html

[30]http://startuplessonslearned.blogspot.com/2008/09/waves-of-technology-platforms.html

[31]http://startuplessonslearned.blogspot.com/2008/09/ideas-code-data-implement-measure-learn.html

[32]http://startuplessonslearned.blogspot.com/2008/09/how-to-listen-to-customers-and-not-just.html

[33]http://startuplessonslearned.blogspot.com/2008/09/one-line-split-test-or-how-to-ab-all.html

---

Let's start with the "do whatever customers say, no matter what" problem. I'll borrow this example from randomwalker's journal - Lessons from the failure of Livejournal: when NOT to listen to your users[34].

> The opportunity was just mind-bogglingly huge. But none of that happened. The site hung on to its design philosophy of being an island cut off from the rest of the Web, and paid the price. ... The site is now a sad footnote in the history of Social Networking Services. How did they do it? **By listening to their users**.

randomwalker identifies four specific ways in which LJ's listening caused them problems, and they are all variations on a theme: listening to the wrong users. The early adopters of LiveJournal didn't want to see the site become mainstream, and the team didn't find a way to stand up for their business or vision.

I remember having this problem when I first got the "listening to customers" religion. I felt we should just talk to as many customers as possible, and do whatever they say. But that is a bad idea. It confuses the tactic, which is listening, with the strategy, which is learning. Talking to customers is important because it helps us deal in facts about the world as it is today. If we're going to build a product, we need to have a sense of who will use it. If we're going to change a features, we need to know how our existing customers will react. If we're working

[34]http://arvindn.livejournal.com/96382.html

on positioning[35] for our product, we need to know what is in the mind of our prospects today.

If your team is struggling with customer feedback, you may find this mantra helpful. Seek out a synthesis that incorporates both the feedback you are hearing plus your own vision. Any path that leaves out one aspect or the other is probably wrong. Have faith that this synthesis is greater than the sum of its parts. If you can't find a synthesis position that works for your customers and for your business, it either means you're not trying hard enough or your business is in trouble. Figure out which one it is, have a heart-to-heart with your team, and make some serious changes.

Especially for us introverted engineering types, there is one major drawback to talking to customers: it's messy. Customers are living breathing complex people, with their own drama and issues. When they talk to you, it can be overwhelming to sort through all that irrelevant data to capture the nuggets of wisdom that are key to learning. In a perfect world, we'd all have the courage and stamina to perservere, and implement a complete Ideas-Code-Data[36] rapid learning loop. But in reality, we sometimes fall back on inadequate shortcuts. One of those is an over-emphasis on split-testing.

Split-testing provides objective facts about our product and customers, and this has strong appeal to the science-oriented

[35]http://www.amazon.com/gp/product/0071359168?ie=UTF8&tag=lessolearn01-20&linkCode=as2&camp=1789&creative=9325&creativeASIN=0071359168

[36]http://startuplessonslearned.blogspot.com/2008/09/ideas-code-data-implement-measure-learn.html

among us. But the thing to remember about split-testing is that it is always retrospective - it can only give you facts about the past. Split-testing is completely useless in telling you what to do next. Now, to make good decisions, it's helpful to have historical data about what has and hasn't worked in the past. If you take it too far, though, you can lose the creative spark that is also key to learning.

For example, I have often fallen into the trap of wanting to optimize the heck out of one single variable in our business. One time, I became completely enamored with Influence: The Psychology of Persuasion[37] (which is a great book, but that's for another post). I managed to convince myself that the solution to all of our company's problems were contained in that book, and that if we just faithfully executed a marketing campaign around the principles therein, we'd solve everything. I convinced a team to give this a try, and they did tried dozens of split-test experiments, each around a different principle or combination of principles. We tried and tried to boost our conversion numbers, each time analyzing what worked and what didn't, and iterating. We were excited by each new discovery, and each iteration we managed to move the conversion needle a little bit more. Here was the problem: the total impact we were having was miniscule. It turns out that we were not really addressing the core problem (which had nothing to do with persuasion). So although we felt we were making progress, and even though we were moving numbers on a spreadsheet, it was all for nothing. Only when someone hit me over the head and said "this isn't working, let's try a radically new direction" did I realize what had happened. We'd forgotten to use the all the tools in our

[37]http://www.amazon.com/gp/product/006124189X?ie=UTF8&tag=lessolearn01-20&link_code=as3&camp=211189&creative=373489&creativeASIN=006124189X

toolbox, and lost sight of our overarching goal.

It's important to be open to hearing new ideas, especially when the ideas you're working on are split-testing poorly. That's not to say you should give up right away, but always take a moment to step back and ask yourself if your current path is making progress. It might be time to reshuffle the deck and try again.

Just don't forget to subject the radical new idea to split-testing too. It might be even worse than what you're doing right now.

So, both split-testing and customer feedback have their drawbacks. What can you do about it? There are a few ideas I have found generally helpful:

- Identify where the "learning block" is. For example, think of the phases of the synthesis framework: collecting feedback, processing and understanding it, choosing a new course of action. If you're not getting the results you want, probably it's because one of those phases is blocked. For example, I've had the opportunity to work with a brilliant product person who had an incredible talent at rationalization. Once he got the "customer feedback" religion, I noticed this pattern: "Guys! I've just conducted three customer focus groups, and, incredibly, the customers really want us to build the feature I've been telling you about for a month." No matter what the input, he'd come around to the same conclusion as before.

Or maybe you have someone on your team that's just not processing: "Customers say they want X, so that's what we're building.” Each new customer that walks in the door wants a different X, so we keep changing direction.

Or consider my favorite of all: the “we have no choice but to stay the course” pessimist. For this person, there’s always some reason why what we’re learning about customers can’t help. We’re doomed! For example, we simply cannot make the changes we need because we’ve already promised something to partners. Or the press. Or to some passionate customers. Or to our team. Whoever it is, we just can’t go back on our promise, it’d be too painful. So we have to roll the dice with what we’re working on now, even if we all agree it’s not our best shot at success.

Wherever the blockage is happening, by identifying it you can work on fixing it.

- Focus on “minimum feature set” whenever processing feedback. It’s all too easy to put together a spec that contains every feature that every customer has ever asked for. That’s not a challenge. The hard part is to figure out the fewest possible features that could possibly accomplish your company’s goals. If you ever have the opportunity to remove a feature without impacting the customer experience or business metrics - do it.

- Consider whether the company is experiencing a phase-change that might make what’s made you successful in the past obsolete. The most famous of these phase-change theories is Crossing the Chasm[38], which gives very clear

[38]http://www.amazon.com/gp/product/0060517123?ie=UTF8&tag=lessolearn01-20&link_code=as3&camp=211189&creative=373489&creativeASIN=0060517123

> guidance about what to do in a situation where you can't seem to make any more progress with the early-adopter customers you have. That's a good time to change course. One possibility: try segmenting your customers into a few archetypes, and see if any of those sounds more promising than another. Even if one archetype currently dominates your customer base, would it be more promising to pursue a different one?

As much as we try to incorporate scientific product development[39] into our work, the fact remains that business is not a science. I think Drucker said it best[40]. It's pretty easy to deliver results in the short term or the long term. It's pretty easy to optimize our business to serve one of employees, customers or shareholders. But it's incredibly hard to balance the needs of all three stakeholders over both the short and long-term time horizon. That's what business is designed to do. By learning to find a synthesis between our customers and our vision, we can make a meaningful contribution to that goal.

The App Store after the gold rush

I wrote earlier about the issue of distribution advantage on the iPhone[41]. As the gold rush drives thousands of apps onto the platform, it's getting harder and harder for new entrants to get

[39]http://startuplessonslearned.blogspot.com/2008/09/thoughts-on-scientific-product.html

[40]http://www.amazon.com/gp/product/0060878975?ie=UTF8&tag=lessolearn01-20&linkCode=as2&camp=1789&creative=9325&creativeASIN=0060878975

[41]http://startuplessonslearned.blogspot.com/2008/09/how-to-get-distrubtion-advantage-on.html

oxygen. Take a look at The App Store after the gold rush - FierceDeveloper[42]:

> According to a recent *BusinessWeek* feature[43], the flood of new games, productivity tools and related iPhone software is making it difficult for the vast majority of apps to crack the consumer consciousness. A number of developers are slashing their prices to remain competitive, but it appears that the gold rush that followed on the heels of the App Store's July 10 grand opening is already over, and the get-rich-quick stories of developers like Steve Demeter–who reportedly raked in $250,000 in just two months for his iPhone game Trism–have already passed into coder lore.

The App Store is a channel for customer acquisition. As the channel gets more and more crowded, just launching an app in the store is getting worse and worse as a strategy for each new entrant. This is completely analogous to the situation elsewhere on the internet, where launching a new website, product, or service with PR is getting harder and harder. Customers and prospects are overwhelmed by the number of media and companies clamoring for their attention. If your launch is not immediately successful, you quickly fall into oblivion. On the App Store, the same dynamic is in play. If your app doesn't immediately make it into the Top 25 page, it's pretty hard to have any kind of durable growth.

[42]http://www.fiercedeveloper.com/story/app-store-after-gold-rush/2008-10-06?utm_medium=nl&utm_source=internal&cmp-id=EMC-NL-FD&dest=FD

[43]http://www.technewsworld.com/story/it-management/64675.html?wlc=1223321245

So what can you do? I think it's helpful to think about two kinds of competition for distribution: acquisition competition and retention competition.

Acqusition competition is how new apps get new customers. On the web, we have many of these channels: SEM, SEO, world of mouth, PR and viral. On the iPhone, it seems that two are driving most of the installs: the "newest apps" RSS feed (which may be combined with PR) and a primitive form of SEO, when people search the App Store for a specific kind of app. Over time, we should get more channels that service the long tail of apps for which the current channels are not working. For example, if any of the mobile ad networks gets major traction, they may become a dominant way that people discover new apps.

Retention competition is how you get people to come back to your app. The primary place this competition is visible is on the home screen of the iPhone itself. But the real battle is in the mind of the people who have installed your application. What causes them to come back to your app, instead of spending their time doing something else? Are they turning on their phone specifically to get your app? Do they browse around looking for an app to pass time? Does your app solve a specific problem that they have? Do you have a way to notify them by SMS or email when something notable happens?

The reason I think it's important to think about retention competition when you are thinking about acquisition is that it strongly influences your acquisition options. If your app has incredibly strong retention, you will probably do very well with the current PR/new app system of acquisition. Why? Because you'll be able to leverage your strong retention to stay in the Top 25 list, which will lead to strong acquisition, in a nice positive feedback loop.

If your app has strong word-of-mouth or viral components, your retention drives new acquisition, and it's not so important to have good placement in the store. If and when a good SEM solution shows up for iPhone, you may be able to use it to artificially drive your app into the Top 25, as a one-time event. Then, if your retention is good enough, you can stay there. Or if your lifetime value is high enough, you can just keep spending on SEM.

So if you have a new app that you are thinking of launching, what should you do? My advice: don't launch big. Don't do PR upfront, don't put out a press release. Figure out how to launch quietly, so you can find out what your retention and referral rates are going to be. If necessary, consider doing this under a different brand name than the one you are wedded to using. Having that data will let you pick an acquisition strategy that is appropriate for your app. It's like knowing the future.

Three decisions to make on virtual goods

I was invited to the Virtual Goods Summit[44] yesterday, and got to see quite a few interesting speakers and panels. It got me thinking about what decisions are essential when building a virtual goods product.

I'm really proud of the virtual goods system we built at IMVU[45],

[44]http://vgsummit2008.com/
[45]http://www.imvu.com/

and a whole day of presentations left me feeling very confident in IMVU's durable competitive advantage. Most of the breakthroughs we had at IMVU were made possible because we had really good people grappling with really difficult problems. Those problems were the necessary consequences of some decisions we made early on. I'd love to take credit for the good work that came later, but that's not really fair. The credit is due to much smarter people than me, and to the incredible power of necessity, that mother of invention.

Here's what I mean. I believe there are only a few key decisions to make about virtual goods. These decisions cause problems that will challenge you for years, if you are lucky enough to be successful with your product. If you know what you're getting into, you really don't need to solve those problems in advance. As the size of your community and economy grows, you'll be able to solve them as they arise, as long as you know what to look for.

What makes a particular decision key? Those that require extreme focus and which are hard to reverse later. Of course, no choices in the real world are completely binary. At IMVU, where we are known for our incredible user-generated content, we also have some first-party content, that was important in seeding our catalog. But the work we've done to enable UGC dwarfs the effort we've put into first-party content, so much so that I think it has rightfully consumed almost all of our focus. Hence, this caveat. I recommend that you focus on one answer to each of these questions, even if you dabble in the other possibility.

User-generated content (UGC) or first-party content?
Some products, like Habbo Hotel or World of Warcraft, employ professional artists and designers to create compelling virtual

goods that players want to buy. Others, like IMVU or Second Life, rely almost exclusively on third-party developers to create goods.

If you go the UGC route, be prepared to learn an awful lot about copyright and trademark law. Be prepared to deal with excruciating judgements about what kind of content is appropriate for your audience. Your challenge will be incentivizing your developers to create, and managing the volume of their output. How do you help customers find the best of the best? And what do you do with the rest? You may have to staff up a large customer service department to review and approve all of the items. And you may find yourself at the whim of some pretty unstable people. Yes, I have had to deal with developers on strike. Are you ready for that?

If you go the first-party route, you can try a different set of problems. You do get the beneift of total control of your content, so you can guarantee your customers a quality experience. However, virtual goods is a hits-driven business, which means you must constantly figure out what kind of content your customers want. Sales depend on being right (whereas in UGC we can pretty much try anything, and promote the winners). Your platform may require you to staff up a large production department to crank out enough content to satisfy your customers. You may have to take template-based shortcuts ("500 colors of the same basic shirt"). You may also struggle to keep up with changes in your audience. If your early adopters are interested in one kind of thing, but your mainstream audience has a different set of tastes, you may find yourself with a lot of rework on your hands. Can your platform really accomodate emo and anime? Princess and goth?

One last note on UGC vs first-party content, especially for you who are thinking of going the first-party route. Be careful not to get yourself accidentally into the UGC business. Humans are incredibly resourceful, and they really like to express themselves. Beware the trap that Google Lively fell into[46]. They thought they had UGC under control, because they prohibited any user-generated 3D content or animations. They did allow people to create chat rooms and set the topic to whatever they wanted. Guess what they wanted to talk about. Lively's launch was marred by zillions of sex chat rooms, with people spending hours and hours trying to figure out how to use Lively's stock animations to make their avatars do things that looked kinky.

Subscription or a la carte payments?
I notice that people sometimes forget that one of the most profitable virtual goods businesses in the world is Blizzard's World of Warcraft, even though they don't allow customers to trade their hard-earned cash for virtual goods directly. Instead, they charge a flat-rate subscription, and then manage goods using a combination of gameplay, an internal currency and various trading systems. Most systems that come out of the video game world use subscriptions, as do products targeted primarily at kids. Why? Subscriptions help keep gameplay balanced between those with lots of money and those with less, which is key for products focused on fun. Subscriptions also constrain payment-method choices. In the US, it generally means getting parents involved, which is great for kid-focused products (that need parents anyway) but not necessarily great for products targeted at teenagers. On the other hand, they have a huge advantage when it comes to chargebacks, because regular

[46]http://www.theregister.co.uk/2008/07/09/google_lively/

subscribers provide a huge base of “safe” transactions that form a stable denominator for chargeback ratios.

Other sites, like IMVU, Mob Wars and many others allow customers to buy items a la carte, one at a time (or in bundles) as an upsell to the standard experience. This is especially important for teenagers, because it allows them to participate in products without having to go through the complex negotiation with their parents that a subscription usually requires. If you support mobile or prepaid cards for billing, they can even do it on their own, without approval, just using their discretionary income. If you go this route, be prepared to make money a dollar at a time, and to work with lots and lots of payment vendors. You’re also going to have to deal with a lot more fraud and chargeback issues, because you’ll have to work harder to forge the kind of long-term billing relationship subscription businesses enjoy. On the other hand, you can service customers that are hard to reach: the unbanked, international customers without credit cards, and teenagers everywhere.

Merchandising or gameplay?
In a game like World of Warcraft, Kart Racer, or Mob Wars, customers don’t struggle to discover new items. It’s obvious what they want to buy, because the objects confer direct benefits that are rooted in the gameplay mechanics. The challenge is to balance the benefits that paying customers get with the benefits that customers get by investing three other attributes: passion, time and skill. An unbalanced game can lose all of its customers (paying or otherwise) rapidly.

For other products, customers are generally shopping for virtual goods in some kind of catalog. If you go this route, be prepared to make a long-term investment in the skill of merchandising.

If you have a non-trivial amount of goods for people to buy, they are going to have to find ways to sort through them to find the ones they want. You'll need bundles, promotions, category and segment managers, and algorithms for upsell, cross-sell, and conversion optimization. In other words, a marketing department.

You can use these three questions to analyze existing businesses. For example, IMVU is a user-generated, a la carte, merchandising product. Habbo is first-party, a la carte, merchandising. Mob Wars is first-party, a la carte, gameplay. WoW is first-party, subscription, gameplay. I hope it also proves useful to people thinking about creating a new virtual goods product. My belief is that many of the decisions you make later will flow from these early ones.

Here's how we thought about it in the early days of IMVU. We first tackled the issue of UGC vs first-party content. This one was easy for us, because our values committed us to try to make freedom of expression core to our product. It was part of our mission statement. Of course, it didn't hurt that we only had a small team and not a lot of money, and third-parties gave us a lot of leverage. As for subscriptions, we wanted to have them from the beginning, but decided against them. Here was our reasoning. Fundamentally, we wanted our third-party developers to make money from their time invested in IMVU. So we did this thought experiment: we imagined a customer who was willing to buy a virtual shirt from one of our developers, but couldn't afford to pay us the subscription fee. Would we really turn them away? We decided the answer was no - our developers interests had to come first. So we chose a la carte. Last, we tackled the gameplay question. We knew IMVU would be a socializing platform, and so we decided that we would let our customers interact using their own rules for assigning status, like we do in the physical world. The government doesn't assign each clothing brand its own level of benefit. The companies that own those brands work hard to differentiate their offering so that people who wear their clothes will be perceived in a certain way. We wanted to let our developers do the same thing, so we opted not to go the gameplay route.

That's how we formed our initial IMVU hypothesis. Very few of the decisions we made in those early days wound up, years later, affecting the outcome of the product. But these three did. I hope sharing them will help others who are thinking of working with virtual goods. If that's you, share your experience in a comment. Did you find these questions helpful? And for those of you already experienced with virtual goods, what do you think is missing?

The engineering manager's lament

I was inspired to write The product manager's lament[47] while meeting with a startup struggling to figure out what had gone wrong with their product development process. When a process is not working, there's usually somebody who feels the pain most acutely, and in that company it was the product manager. Last week, I found myself in a similar situation, but this time talking to the engineering manager. I thought I'd share a little bit about his story.

[47]http://startuplessonslearned.blogspot.com/2008/10/product-managers-lament.html

This engineering manager is a smart guy, and very experienced. He has a good team, and they've shipped a working product to many customers. He's working harder than ever, and so is his team. Yet, they feel like they are falling further and further behind with every release. The more features they ship, the more things that can go wrong. Systems seem to randomly fail, and as soon as they are fixed, a new one falls over.

Even worse, when it comes time to "fix it right" the team gets pushback from the business leaders, who want more features. If engineers want more time to spend making their old code more pretty, they are invited to do so on the weekends.

Unfortunately, the weekends are already taken, because features that used to ship on a friday now routinely cause collateral damage somewhere else on the platform, and so the team is having to work nights and weekends cleaning up after each launch.

Every few months, the situation comes to a head, where the engineers finally scream "enough!" and force the whole company to accept a rewrite of some key system. The idea is that once we move to the new system (or coding standard, or API, or ...) then all the problems will be solved. The current code is spaghetti, but the new code will be elegant. Sometimes the engineers win the argument, and sometimes they are overruled. But it doesn't seem to matter. The rewrite seldom works[48], and even when it does, a few months later they are back in the same dilemna, finding their now-old code falling apart. It's become "legacy code" and part of the problem.

It's tempting to blame the business leaders ("MBA-types") for

[48]http://www.joelonsoftware.com/articles/fog0000000069.html

this mess. And in a mess this big, there is certainly blame to go around. But they are pushing for the things that matter to customers - features. And they are cognizant that their funding is limited, and if they don't find out which features are absolutely critical for their customers soon, they won't be able to survive. So they are legitimately suspicious when, instead of working on adding monetization to their product, some engineer wants to take a few weeks off to go polish code that is supposed to be already done.

What's wrong with this picture? And why is the engineering manager suffering so badly? I can relate to his experience all too well. When I was working my first programming jobs, I was introduced to the following maxim: "time, quality, money - pick two." That was the watchword of our profession, and I was taught to treat with disdain those "suits" who were constantly asking for all three. We treated them like they had some kind of brain defect. If they wanted a high-quality product done fast, why didn't they want to pay for it? And if money was so tight, why were they surprised when we cut corners to get the product out fast? Or went past the deadline to get it done right?

I really believed this mantra, for a time. But it started to smell bad. In one company, we had a QA team as large as our engineering team, dozens of people who worked all day every day to find and report bugs in our prodcut. And this was a huge product, which took years to develop. It was constantly slipping, because we had a hard time adding new features while trying to fix all the bugs that the QA team kept finding. And yet, it was incredibly expensive to have all these QA testers on staff, too. I couldn't see that we were managing to pick even one. Other, more veteran programmers told me they had seen this in many companies too. They just assumed it was the way software companies worked.

Suffice to say, I no longer believe this.

In teams that follow the "pick two" agenda, which two has to be resolved via a power play. In companies with a strong engineering culture, the engineers pick quality. It's their professional pride on the line, after all. So they insist on having the final say on when a feature is "done" enough to show to customers. Business people may want to speed things up by spending more money, but enough people have read the Mythical Man-Month[49] to know that doesn't work.

In teams that have a business culture, the MBA's pick time. After all, our startup is on a fixed budget. They set deadlines, schedules, and launch plans, and expect the engineering team to do what it takes to hit them. If quality suffers, that's just the way it is. Or, if they care a lot about quality, they will replace anyone who ships without quality. Unfortunately, threats work a lot better at incentivizing people to CYA[50] than getting them to write quality software.

A situation where one faction "wins" at another's expense is seldom conducive to business success. As I evolved my thinking, I started to frame the problem this way: How can we devise a product development process that allows the business leaders to take responsibility for the outcome by making conscious tradeoffs?

When I first encountered agile software techniques, in the form

[49]http://www.amazon.com/gp/product/0201835959?ie=UTF8&tag=lessolearn01-20&link_code=as3&camp=211189&creative=373489&creativeASIN=0201835959

[50]http://en.wikipedia.org/wiki/Cover_your_ass

of extreme programming[51], I thought I had found the answer. I explained it to people this way: agile lets you make the trade-offs visible to whole company, so that they can make informed choices. How? By shipping software early, you give them continuous feedback about how it well it's working. They can use the software themselves, since every iteration produces working (if incomplete) code. And if they want to invest in higher quality, they can. But, if they want to invest in more experiments (features), they can do that too. But in neither case should they be surprised by the result. Sound good?

It didn't work. The business leaders I've run this system with ran into the same traps as I had in previous jobs. I had just passed the burden on to them. But of course they didn't feel reponsible for the outcome - that was my job. So I wound up with the worst of both worlds: handing the steering wheel over to someone else, but then still being blamed for the bad results.

Even worse, agile wasn't really helping me ship higher quality software. We were using it to get features to market faster, and that was working well. But we were cutting corners in the development methodology as well as in the code, in the name of increased speed. But because we had to spend more and more time fixing things, we started slowing down, even as we tried to speed up. That's the same pain the engineering manager I met with was experiencing. As the situation deteriorates, he's got to work harder and harder just to keep the product from regressing.

It was my own failure to ship quality software in the early days of IMVU[52] that really got me thinking about this problem in a new way. I now believe that the "pick two" concept is

[51]http://extremeprogramming.org/

[52]http://www.imvu.com/

fundamentally flawed, and that lean startups[53] can achieve all three simultaneously: quickly bring high-quality software to market at low cost. Here's why.

First of all, it's a myth that cutting corners saves time. When we ship software with defects, we wind up having to waste time later dealing with those defects. If each new feature contains a few recurring problems, then over time we'll become swamped with the overhead of fixing and won't be able to ship any new features.

So far, that sounds like just another argument for "doing things right" the first time, no matter how long they take. But that's problematic too. The biggest form of waste is building software nobody wants. The second biggest form of waste is fixing bugs in software nobody wants. If we defer fixing bugs in order to bring a product to market sooner, and this allows us to find out if we're on the right track or not, then it was worthwhile to ship with those bugs.

Here's how I've resolved the paradox in my own thinking. There are two kinds of bugs:

- One kind are what I call defects: situations where the software doesn't behave in a predictable way. Examples: intermittently failing code, obfuscated code that's hard to use properly, code that's not under test coverage (and so you don't know what it does), bad failure handling, etc.
- The second kind of bugs are the type your traditional QA tester will find: situations where the software doesn't do what the customer expects. For example, you click

[53]http://startuplessonslearned.blogspot.com/2008/09/lean-startup.html

> on a button labeled “Quit” and in response the software erases your hard drive. That’s a bug. But if the software reliably erases your hard drive every time, that’s not a defect. The resolution to the paradox is to realize that only defects cause you future headaches, and cannot be deferred. That’s why we need continuous integration and test-driven development. Whenever we add a feature or fix a bug, we need to make sure that code never goes bad, never mysteriously stops working. Those are the kinds of indefinite costs that make our team grind to a halt over time. Traditional bugs don’t - you can choose to fix them or not, depending on what your team is trying to accomplish.

Defects are what make refactoring[54] difficult. When you improve code, you always test it at least a little bit. If what you see is what will ultimately make it into production, it’s pretty easy to make sure you did a good job. But code that is riddled with defects is a cameleon - one moment it works, the next it doesn’t anymore. That leads to fear of refactoring, which leads to spaghetti code.

By shipping code without defects, the team actually speeds up over time. Why? Because we never have to revisit old code to see why it stopped working. We can add new team members, and they can get started right away (as an aside, new engineers at IMVU were always required to ship something to customers on their first or second day). And the whole team is gettng smarter and learning, so our effectiveness increases. Plus, we get the benefit of code reuse, and all the great libraries and tools we’ve built. Every iteration, we get a little more done.

[54]http://www.amazon.com/Refactoring-Improving-Existing-Addison-Wesley-Technology/dp/0201485672

So how can I help the engineering manager in pain? Here’s my diagnosis of his problem:

- He has some automated tests, but his team doesn’t have a continuous integration server or practice TDD. Hence, the tests tend to go stale, or are themselves intermittent.
- No amount of fixing is making any difference, because the fixes aren’t pinned in place by tests, so they get dwarfed by the new defects being introduced with new features. It’s a treadmill situation - they have to run faster and faster just to stay at the level of quality/features they’re at today.
- The team can’t get permission from the business leaders to get “extra time” for fixing. This is because the are constantly telling them that features are done as soon as they can see them in the product. Because there are no tests for new features (or operational alerts for the production code), the code that supports those new features could go bad at any moment. If the business leaders were told “this feature is done, but only for an indeterminate amount of time, after which it may stop working suddenly” they would not be so eager to move on to the next new feature.

Here’s what I’ve counseled him to try:

- Get started with continuous integration. Start with just one test, a good one that runs reliably, and make sure it gets run on every checkin.

- Tie the continuous integration server in with source control, so that nobody can check in while the tests are failing.
- Practice five why's to get to the root cause of future problems. Use those opportunities to add tests or alerts that would have prevented that problem. Make the investment proportional to the problem caused, so that everyone (even the business leaders) feels good about it.

My prediction is that these three practices will quickly change the feeling around the office, because the most important code will wind up under test quite soon (after all, it's the code that people care the most about, and so when it fails, the team notices and fixes it right away). With the most important code under test, the level of panic when something goes wrong will start to decrease (because it will tend to be in less important parts of the product). And has the tension goes down, it will be easier to get the whole team (including the MBA's) to embrace TDD and other good practices as further refinements.

Good luck, engineering manager. May your team, one day soon, refactor with pride.

Lean startups vs lean companies

Venture Hacks[55] has a great article today[56] on lean startups[57]. They quote extensively from two of our most important thinkers:

[55]http://venturehacks.com/

[56]http://venturehacks.com/articles/lean-startups

[57]http://startuplessonslearned.blogspot.com/2008/09/lean-startup.html

Taiichi Ohno[58], creator of the Toyota Production System, and Kent Beck[59], creator of extreme programming. If you're not familiar with their work, and how it relates to startups, this is a great place to start. Go read it[60].

I want to take up one of the questions posed in the Hacker News discussion[61] of the article:

> The question in my mind against most of these efficiency driven programs is simple. If you're always looking to eliminate waste and become super efficient, you're not spending any time being creative or chasing radical ideas that may or may not be worth the effort spent on executing them.
>
> Innovation sometimes requires a lot of wasteful experimentation and it looks like that is completely anti-thetical to the whole efficiency argument.

This is a common sentiment that derails many efficiency programs. But I do think it is a misconception of what lean startups are all about. Part of the problem is the distinction (that I owe to Steve Blank[62]) between lean startups and lean companies. In a typical lean company, waste is defined as "every activity that does not create value for the customer." And this is 100%

[58]http://www.amazon.com/Toyota-Production-System-Beyond-Large-Scale/dp/0915299143/ref=sr_1_1?ie=UTF8&s=books&qid=1224646274&sr=8-1

[59]http://www.amazon.com/Extreme-Programming-Explained-Embrace-Change/dp/0321278658/ref=sr_1_3?ie=UTF8&s=books&qid=1224646295&sr=1-3

[60]http://venturehacks.com/articles/lean-startups

[61]http://news.ycombinator.com/item?id=339430

[62]http://www.amazon.com/Four-Steps-Epiphany-Steven-Blank/dp/0976470705/ref=sr_1_1?ie=UTF8&s=books&qid=1224646351&sr=1-1

correct. By driving this kind of waste out of your company, you actually boost creativity by eliminating beaurocracy, busy work, unnecessary hierarchy, and, of course, excess inventory.

But statups require a special kind of creativity: disruptive innovation. And, as the commenter rightly points out, this is not really a matter of efficiency. By the standard of "customer value," most innovation-seeking experiments are waste. Lean startups operate by a different standard. I suggest they define waste as "every activity that does not contribute to learning about customers." (aka "how you get to product/market fit[63].") This is why I find the concept of the Ideas-Code-Data feedback loop[64] so helpful. Any activity that actively promotes us getting through each iteration (including the learning phase) faster, is value-creating. Everything else (including a lot of traditional "agile" or "lean" tactics) is waste.

Of course, lean startups and lean companies both follow the same underlying principles. It's just the implementation that changes, as focus shifts from learning to execution. And, as many commenters noted, there are great companies that successfully incorporate innovation into their execution discipline (Toyota first among them). I would say that the true standard of waste is "anything that does not carry out the company's mission." But I think that is too abstract to be really useful for most startups, since it's hard to realize that the mission changes as the company goes through the natural phase changes of growth.

[63]http://venturehacks.com/articles/plans-ndas-traction#traction

[64]http://startuplessonslearned.blogspot.com/2008/09/ideas-code-data-implement-measure-learn.html

Chuck's Code & Learning: Learning to write tests that matter

Chuck's Code & Learning: Learning to write tests that matter[65]

> Don't write tests for scenarios just because you can.
> Write tests that support a specific business need.

Amen.

A hierarchy of pitches

Every company will need to pitch itself from time to time. Usually we think of pitches in the context of raising money, but that is only one of many pitch situations. We pitch to potential partners, vendors, publishers, conferences, employees, and even lawyers. It's different from selling a product, because it is not part of our regular business practice, is not something that relates to our core competence, and tends not to happen in a repeatable and scalable way. (I'll exclude those non-lean startups[66] who basically exist for the purpose of raising bigger and bigger sums of money. You're not one of those are you?)

Most of the times I have seen pitches fail, it is not because they are poorly written, or that the entrepreneur lacks passion. It is because they don't answer the right question. My favorite example of all time comes from students in an entrepreneurship

[65]http://choosetheforce.blogspot.com/2008/10/learning-to-write-tests-that-matter.html

[66]http://startuplessonslearned.blogspot.com/search/label/lean%20startup

class. Their idea was to build a next-generation autonomous robot, that could be used by defense and security agencies around the world. The whole pitch was about how valuable robots could be in the future. They even included a slide with The Transformers on it. Now there was nothing wrong with their analysis: anyone who invents a technology as sophisticated as The Transformers is definitely going to make a lot of money. But these students completely failed to address the one and only question on their audience's mind: can you three guys really build the robots of the future? (Turns out, they were incredibly well-credentialed graduate students who had, in fact, developed some interesting new robotics technology. But you wouldn't have known that from their pitch.)

I have come to believe that there is a hierarchy of pitches, and that understanding where your pitch falls on this spectrum can assist in making decisions about what information to highlight. Pitches higher in the hierarchy tend to be more successful, and so if you can fit your company into one of those categories, you can get better results or better terms. Now, just because you can do a thing, doesn't mean you should - and there are plenty of other great resources[67] out there that can help you think through whether and when to raise money (or do other kinds of deals).

With that disclaimer out of the way, here's how I order the hierarchy of pitches:

Printing money
Key questions: are those numbers real? how big is the market? can your team execute the growth plan?
Most important slide: valuation

[67]http://startuplessonslearned.blogspot.com/2008/09/paul-graham-on-fundraising.html

Promising results
Key questions: can you monetize that traffic? (or drive traffic to that profitable destination?) do you know why you've achieved those results?
Most important slide: hockey stick

Micro-scale results
Key questions: who is the customer, and how do you know? what is the potential market size? what are the business economics?
Most important slide: lessons learned

Working product
Key questions: what does the product do? what's the launch plan? who's on the marketing team?
Most important slide: live demo

Prototype product
Key questions: what will it take to ship a working product? how do you know anyone would want it? who's on the engineering team?
Most important slide: demo (if the product solves an obvious problem), engineering resumes (if the product is nearly impossible to build), "day in the life of a customer[68]" (if neither of the above)

Breakthrough technology
Key questions: who owns the patents? can we make a product out of this technology? are there any good substitutes?
Most important slide: barriers to entry

[68]http://books.google.com/books?id=oLL2pjn2RV0C&pg=PA40&lpg=PA40&dq=steve+blank+%22day+in+the+life+of+a+customer%22&source=web&ots=ucRoMlQ-Dz&sig=O_J5QJn5yf-_DcdkN154r6txPec&hl=en&sa=X&oi=book_result&resnum=2&ct=result

All-star team
Key questions: has the team made money for their investors in the past? are they domain experts? are they committed to an idea in their domain of expertise?
Most important slide: problem we are trying to solve

Good product idea
Key questions: what kinds of risk does this company need to mitigate (technology risk, market risk, team risk, funding risk)? is it a revolutionary and novel idea? is this team the one to back? can the team bring the product to market? who is the customer? who is the competition? will they fail fast?
Most important slide: about the founders

In a pitch meeting, try to spend as much time as possible talking about the key questions for your pitch. If you find yourself getting asked non-key questions, try to use your answers to steer the conversation back to the key questions. But here's the most important part: if you keep getting non-key questions over and over again, something is wrong with your pitch. Either you misunderstand where your pitch fits into the hierarchy, or you are not using the early part of your pitch to establish it. Don't keep banging your head against the wall - if you can't convince your potential partners that your startup is printing money, try to figure out why. Experiment with different narratives. If you still can't do it, move one level down the hierarchy and see if you can make that story stick.

One last piece of advice: don't forget that potential partners are evaluating the strength of your pitch, not you. It's not true that companies with pitches further up the hierarchy are better, in some absolute sense, than companies further down. It's just that they have an easier time closing these kinds of deals. If you can't close the deal, maybe your company is at the wrong stage of its development, and it's time to try a different tack.

John Doerr's 10 lean startup tips

I just saw video of John Doerr's talk yesterday at VentureBeat's "How to manage your start-up in the downturn" roundtable event[69]. The tips are based on advice JD solicited from great KPCB entrepreneurs. I was impressed enough to transcribe (and paraphrase) the list. You can see the whole video at the end of this post.

1. Act now, act with speed.
2. Protect the vital core of the business.Use a scalpel to make strategic cuts.
3. Get 18 months or more of cash (runway) in the business against a conservative forecast.
4. Defer all facilities expansions, capital expenditures. Use Google Apps. Reprioritize & rerationalize all R&D. Defer.
5. Negotiate. Everything is negotiable in this climate.
6. Everybody should be selling. Selling is an honorable profession. Everyone from the receptionist to engineers is selling. Not just about expenses, about increasing revenue.

[69]http://kara.allthingsd.com/20081028/how-to-manage-your-start-up-in-the-downturn-well-come-to-this-event-and-find-out/

7. Offer equity instead of cash. Voluntary salary reduction program.
8. Pay attention to where your cash is. All cash in most secure possible instruments.
9. Make sure for planned revenues you have "leading indicators" to know if you will hit it.
10. Over-communicate with employees, investors, customers. Don't sugar coat.

What I find remarkable about this list is how many of them apply to lean startups[70] in good times and in bad. For example, on the "leading indicators" front, I think companies should use a funnel analysis to drive decisions and get rapid feedback on how they are progressing. This can take the form of a traditional sales pipeline or a registration-activation-revenue[71] chart. I've also used the voluntary salary reduction tool - it's particularly useful as a way to get passionate employees who don't have a lot of personal expenses to buy into the mission of the company at a deep level. And even for those employees who can't afford to take much reduction, it's sobering to go through the exercise of deciding if they believe in the company enough to put their own money into it. At IMVU[72] we strongly believed that everyone in the company had to be involved in selling[73]. There's no better

[70]http://startuplessonslearned.blogspot.com/2008/09/lean-startup-comes-to-stanford.html

[71]http://startuplessonslearned.blogspot.com/2008/09/three-drivers-of-growth-for-your.html

[72]http://www.imvu.com/

[73]http://startuplessonslearned.blogspot.com/2008/09/sem-on-five-dollars-day.html

way to learn what customers want[74] (or hate!). And it's hard to know if you're truly succeeding without a focus on revenue.

I hope the startups who are struggling with the current downturn will use it as a motivator to make cuts that actually increase their tempo and speed. A crisis can clarify what's important, and getting clear about what's important is the criticial first step to seeing waste clearly (see Lean Thinking[75]). And once you can see waste, you can start to get it out of the way of your execution, and become a truly lean company[76].

The Entire Video of John Doerr Giving 10 Tips for Start-ups to Avoid the Econalypse[77]

Related articles by Zemanta

- Ten ways to save your company by John Doerr[78]
- John Doerr's 10 Ways to Stay on Top[79]
- Ten ways to save your company by John Doerr[80]
- John Doerr: 10 ways for companies to stay afloat in rough times (MG Siegler/VentureBeat)[81]

[74]http://startuplessonslearned.blogspot.com/search/label/listening%20to%20customers

[75]http://www.amazon.com/gp/product/0684810352?ie=UTF8&tag=lessolearn01-20&link_code=as3&camp=211189&creative=373489&creativeASIN=0684810352

[76]http://startuplessonslearned.blogspot.com/2008/10/lean-startups-vs-lean-companies.html

[77]http://kara.allthingsd.com/20081030/the-entire-video-of-john-doerr-giving-10-tips-for-start-ups-to-avoid-the-econalypse/

[78]http://myventurepad.com/MVP/39484

[79]http://www.killerstartups.com/blog/john-doerr%25e2%2580%2599s-10-ways-to-stay-on-top/

[80]http://www.texasstartupblog.com/2008/10/29/ten-ways-to-save-your-company-by-john-doerr/

[81]http://www.techmeme.com/081029/p60

July 2010

The Entrepreneur's Guide to Customer Development

Brant Cooper and Patrick Vlaskovits have written a new book, *The Entrepreneur's Guide to Customer Development*[82], which builds upon the foundational work of *The Four Steps to the Epiphany*[83], while improving accessibility, updating the ideas, and making it more actionable. I believe it is the best introduction to Customer Development you can buy.

As all of you know, Steve Blank[84] is the progenitor of Customer Development[85] and author of *The Four Steps to the Epiphany*[86]. I have personally sold many copies of his book, and continue to recommend it as one of the most important books a startup founder can read.

I used to give copies of *Four Steps* out to my employees, in the hopes that it would instantly indoctrinate them into the methodology of Customer Development. I just assumed that everybody would love the book as much as I did, and would instantly change their behavior based on what they read in a book. You can imagine how well that worked. Instead of that

[82]http://www.custdev.com/

[83]http://www.amazon.com/Four-Steps-Epiphany-Steven-Blank/dp/0976470705?ie=UTF8&tag=lessolearn01-20&link_code=btl&camp=213689&creative=392969

[84]http://steveblank.com/

[85]http://www.startuplessonslearned.com/2008/11/what-is-customer-development.html

[86]http://www.amazon.com/Four-Steps-Epiphany-Steven-Blank/dp/0976470705?ie=UTF8&tag=lessolearn01-20&link_code=btl&camp=213689&creative=392969

naive approach, I wish I'd had a book like this one, to help me figure out how to get started with customer development step-by-step.

When I wrote a review of [87]_Four Steps[88] _on this blog in November, 2008, I did my best to be candid and warn of a few shortcomings:

> And Steve is the first to admit that it's a "turgid" read, without a great deal of narrative flow. It's part workbook, part war story compendium, part theoretical treatise, and part manifesto. It's trying to do way too many things at once. On the plus side, that means it's a great deal. On the minus side, that has made it a wee bit hard to understand.

Brant and Patrick undertook a difficult challenge: to provide a generally accessible introduction to Customer Development, without diluting its impact or dumbing-down its principles. I think they've succeeded.

The Entrepreneur's Guide is an easy read. It is written in a conversational tone, doesn't take itself too seriously, and avoids extraneous fluff. It does a great job of laying out general principles and suggesting specific, highly actionable tactics. You can easily take from it whatever makes sense for your business, and leave the rest. And it's incredibly to-the-point: you can digest this book in a couple of hours.

[87]http://www.blogger.com/goog_787136929

[88]http://www.startuplessonslearned.com/2008/11/what-is-customer-development.html

While the customer development framework of *Four Steps* is universally relevant, *The Entrepreneur's Guide* updates its practices for modern startups. _Four Steps _primarily centers its stories and case studies on B2B hardware and software startups. This new volume also tackles examples from the Internet and wireless startups of today, both B2B and B2C. And throughout, they maintain a thoroughly realistic take on the power - and limitations - of an entrepreneurship methodology:

> Successful implementation of Customer Development, let alone simply believing in it, will not guarantee success for your business. Customer Development will help you – force you – to make better decisions based on tested hypotheses, rather than untested assumptions. The results of the Customer Development process may indicate that the assumptions about your product, your customers and your market are all wrong. In fact, they probably will. And then it is your responsibility, as the idea-generator (read: entrepreneur), to interpret the data you have elicited and modify your next set of assumptions to iterate upon.
>
> Many "airport business books" urge entrepreneurs to never give in. They tell them to persist in their dream of building a great product and/or company, no matter what the odds are or what the market might be telling them – success is just around the corner. They tend to illustrate this sort of advice with inspiring stories of entrepreneurs who succeeded against all odds and simply refused to

> throw in the towel. While maintaining persistence and willpower is certainly good advice, Customer Development methodologies are designed to give you data and feedback you may not want to hear. It is incumbent upon you to listen.

The Entrepreneur's Guide to Customer Development includes four powerful case studies/interviews with successful entrepreneurs who have taken iterative approaches to their respective startups that very much resemble the spirit of Lean Startups and Customer Development. I found these to be particularly interesting and worthwhile.

At the heart of Brant and Patrick's interpretation of Customer Development is their belief that its fundamental teaching is to question assumptions. This gives them a hook with which to apply their ideas to a wide variety of situations. In other words, if particular examples in the book don't apply to you directly, Brant and Patrick show you how to figure out what might work for you. This is important, since every situation is different. I'll give them the last word:

> You are already skeptical of Customer Development and Lean Startups and the slew of emerging buzzwords and supple-to-the-point-of-meaningless terms. That's great, more power to you; we applaud your skepticism. But be philosophically consistent: periodically take the time to question your own expertise and that of your friends, partners and investors. Make the effort to test your assumptions.

If there's a shortcoming to this book, it's that it focuses primarily on the Customer Discovery step in *The Four Steps*. Here's hoping they soon tackle Customer Validation. Well done, Brant and Patrick. I can't wait to see what's next. In the meantime, go buy a copy of *The Entrepreneur´s Guide to Customer Development*[89] right now.

Founder personalities and the "first-class man" theory of management

At any given time, something like four percent of the US population is engaged in some form of new-company-creation. And that narrow definition of entrepreneurship doesn't count all of the managers inside established companies who are effectively engaged in the same process of building an internal startup (see What is a startup?[90] for my more expansive definition).

What motivates all these entrepreneurs? Typical explanations tend to focus on the well-known anecdotes and larger than life archetypes we have in mind: the twenty-something college dropouts (men, of course) from Stanford inventing some radical new technology. The academic research tells a very different story.

What do entrepreneurs look like? Are they born or made? This is a hard question. I think the root cause of that difficulty is that we tend to conflate two different questions into one. First, what causes someone to attempt entrepreneurship instead of a

[89]http://www.custdev.com/

[90]http://www.startuplessonslearned.com/2010/06/what-is-startup.html

more traditional career path? And second, what attributes make someone likely to be a successful entrepreneur?

The difficulty lies in this paradox: *many of the attributes that increase the likelihood of becoming an entrepreneur actually impede startup success.*

Let's start with the startup personality attributes. The academic research here is extraordinary. Here are the personality traits that are positively correlated with likelihood to pursue entrepreneurship: extraversion, skepticism, need for achievement, risk taking, desire for independence, locus of control, self efficacy, overconfidence, representativeness (the tendency to over-generalize from small samples), and intuition.

I think most of those factors correspond to our shared image of what an entrepreneur is supposed to look like. But many other attributes (especially demographic realities) cut against that stereotype. For example, Vivek Wadhwa and others[91] have shown that most entrepreneurs are much older than we expect. Career experience and industry expertise are both positively correlated with entrepreneurship: contrary to stereotype, most entrepreneurs are not young and inexperienced outsiders. And unlike some psychological factors, these experience-based factors also increase the odds of the subsequent venture being successful.

It is in the psychological factors that we find the most paradox. For example, consider the propensity for risk-taking. Research has demonstrated the obvious: that people who have greater tolerance for risk or ambiguity are more likely to attempt entrepreneurship. That's not too surprising. But does a risk-taking

[91]http://wadhwa.com/blog/research/

attitude actually lead to more startup success? The studies that have looked at this question in particular have found a negative correlation between risk-taking behavior and startup success.

That doesn't strike me as shocking. And, although this hasn't been subjected to a great deal of study (yet), I believe this same pattern will be found in a variety of other entrepreneurial characteristics: overconfidence, determination to succeed, perseverance, and even the desire to be in control. All of these factors are helpful in getting people to take the plunge, but all of them cause serious impairment of decision-making down the road. Think of the startups you know who are caught in a reality distortion field, heading full-speed off a cliff. Most likely, you will find the above attributes in excess supply.

I believe this is also why breakthrough success stories in entrepreneurship often feature a "classic" zany entrepreneur paired with someone you wouldn't expect to be taking those kinds of risks. We often talk about this as the "visionary" and "the quant" or the "leader" and the "manager." But I'm not convinced those labels are right at all. I think it much more likely that we're seeing the embodiment – in the form of personality - of the "problem team/solution team" organizational structure. One team is in charge of carrying out the vision as currently specified, and one team is constantly asking the skeptical questions: who is the customer? Are we solving the right problem?

Although we have historically viewed this structure in startups by focusing on the personalities of the founders, I think that reflects our current, relatively poor, understanding of how startups work. We can do better by focusing on process instead of personality. We can consciously organize startups to become much more resilient organizations. Otherwise, we risk having them degenerate into cults of personality.

In the early twentieth century, before the advent of scientific management[92], the overriding management philosophy was that of the *first-class man* (and they were always men). The idea was, for any job, if you can simply find an individual with just that right combination of virtues, talents, and experience, you could safely delegate all decisions to them. Sound familiar? This kind of reasoning is almost impossible to disprove. If you empower someone to make decisions and then something goes horribly wrong, does that disprove the *first-class man* theory? Probably not; it's much easier to blame the particular person who made the mistakes. In fact, making mistakes is seen as "proof" of being second-class.

In management jobs related to operations – that is, the people tasked with actually making and distributing physical products – this kind of thinking is now considered ludicrous, thanks to a century of progress. Our modern philosophy of management has this core belief (taken straight from scientific management) at its heart: that the performance of companies is determined by the systems they create, not just the people they hire. No amount of individual superstardom can overcome a badly organized factory, because the weight of the system eventually overwhelms any well-intentioned but poorly organized resistance.

Yet we tolerate our modern version of the first-class man theory in the management of more "fuzzy" topics, especially innovation and entrepreneurship. When we look back on this period in history, it will seem just as ludicrous to future entrepreneurs as

[92]http://www.amazon.com/Principles-Scientific-Management-Frederick-Winslow/dp/1153717824?ie=UTF8&tag=lessolearn01-20&link_code=btl&camp=213689&creative=392969

pre-scientific management looks to us.

I am determined to do everything I can to hasten the arrival of that day. If you're part of the Lean Startup movement, then you're actually making it happen. Thank you.

All of the academic research alluded to in this essay is drawn from Scott Shane's ***General Theory of Entrepreneurship***[93] *which is a fantastic and wide-ranging overview of the state of the art in academic research on entrepreneurship.*

Some IPO speculation

Inspired by Steve Blank's post today about the "lost decade" of IPO's[94], I'd like to make some predictions. Let me be clear: Steve is the historian. His posts are born of tremendous research into the secret history of Silicon Valley[95], and if you haven't read those essays, you should[96]. By contrast, what I'm about to say is pure speculation.

The fact that IPO's are disappearing makes intuitive sense to me. And the fact that the effects of this IPO vanishing act are being felt first and foremost in the software business also makes sense to me. In fact, I believe that the software business is the canary in the coal mine: increasingly, all businesses are going to look more and more like software businesses.

[93]http://www.amazon.com/General-Theory-Entrepreneurship-Individual-opportunity-Horizons/dp/1843769964?ie=UTF8&tag=lessolearn01-20&link_code=btl&camp=213689&creative=392969

[94]http://steveblank.com/2010/07/15/welcome-to-the-lost-decade-for-entrepreneurs-ipos-and-vcs/

[95]http://steveblank.com/category/secret-history-of-silicon-valley/

[96]http://steveblank.com/category/secret-history-of-silicon-valley/

My belief is that the root cause of the IPO shortage is that successful startup companies cannot find productive ways to invest large amounts of money to scale anymore. For software companies especially, scaling distribution and development is comparatively cheap. The old ways aren't working. Large capital investments simply don't have the ROI they used to. The world is changing too fast. New products become commoditized too fast. Increasingly, the only profitable thing to invest in is innovation, which means investing in people. And we don't yet know how to do that on a consistent, scalable, basis.

Ironically, the VC's who depend on IPO's and the CEO's who are supposed to be creating them are struggling with the same basic problem. They do not have a comprehensive theory of entrepreneurship that allows them to consistently invest in innovation that can create long-term value.

The only way I can see to achieve sustained growth is to create an innovation factory. The modern CEO needs to build an organization that is truly diversified: it is continuously investing in successful sustaining innovation and disruptive innovation[97]. Such an organization should be able to deploy large amounts of capital effectively, by investing in its people. But this is a very different kind of diversification from the old-school GE model. We can't just diversify across industries or geographies. We can't even rely on a suite of line-extension products. We have to continually invent new categories of products, new platforms, and new business models – all extremely risky bets. Oh, and by the way, we still have to execute flawlessly. (Even the smallest

[97]http://www.amazon.com/Innovators-Solution-Creating-Sustaining-Successful/dp/1578518520?ie=UTF8&tag=lessolearn01-20&link_code=btl&camp=213689&creative=392969

flaw with an antenna can derail the whole train.)

Today's management reality is just plain harder than that of the past. General managers need to know how to manage the execution-oriented "twentieth century" general managers that work for them. But they also need to know how to manage the new entrepreneurial managers that, increasingly, are essential to their growth. In other words, the old "manager vs. entrepreneur" dichotomy is breaking down. You cannot be a competent general manager in today's economy if you do not understand entrepreneurship.

We are living in a transitional moment. The last of the old-school IPO companies are behind us (at least in software), and yet we have not yet witnessed the new-style IPO companies. Despite Google's reputation as an innovative company, they seem to me to be counted as one of the last of the old breed. Their entrepreneurial successes are mostly to be found outside the organization, in the form of ex-Googlers who became entrepreneurs. Their internal "startup" projects seem, at least to my eye, to have a success rate only marginally higher than Microsoft's (perhaps with Android – an acqusition – as a major exception). Certainly a large proportion of them end in failure.

The new breed of company currently finds itself satisfied with private capital, and has no need for an IPO. I think Zynga and other games companies may be the earliest exemplars for us to look at. Game companies naturally lend themselves to a "studio" model, with semi-autonomous teams building out their own franchise of sequels and spin-offs. From public reports, it seems like Zynga has really mastered a formula for innovating repeatedly and relentlessly across segments, platforms, and genres. And there are plenty of imitators and fast followers on their way. Will that cohort of companies need an IPO?

When private capital is available in sufficient quantities to satisfy investor and founder liquidity needs, why go IPO? The only reason I can think of is when you need dramatically more capital to grow your business, at a magnitude only available on the public markets and therefore worth the loss of control that going public entails. Our leading crop of pre-IPO web companies apparently do not need that much capital – yet.

It's a debatable proposition why they don't need IPO levels of cash. I freely admit that I have no inside information, no unique insight into what they are thinking. But I am nonetheless confident in my prediction: they don't yet have the ability to manage an innovation factory at that scale. That's not a criticism; nobody knows how – yet.

We need a new generation of managers trained in a comprehensive management theory of entrepreneurship. Comprehensive means it has to address all aspects of a startups life: marketing *and* product development, especially. It has to address all stages of startup growth and development – especially including the evolution into a true innovation factory. Tomorrow's managers will need to know how to build a learning organization (where progress is measured by validated learning[98]) and an execution organization (where progress is measured according to traditional value streams). They will need to know how to combine those organizations into one coherent whole.

I believe that the Lean Startup[99] is the first such comprehensive

[98]http://www.startuplessonslearned.com/2009/04/validated-learning-about-customers.html

[99]http://www.startuplessonslearned.com/2010/04/five-myths-about-lean-startup.html

entrepreneurship theory. But these are still early days. We have much work to do. We face problems today that would have bewildered the earliest management theorists. Their struggle was to use management to create enough productivity to feed, clothe, and house the world. Our civilization has excess capacity everywhere. We can build anything we can imagine. But the ranks of highly educated unemployed and the abysmal failure rates of new products both speak to the same question that needs answering: not, "can it be built?" but rather, "should it be built?"

The sum total of all we know about entrepreneurship is just the tip of the iceberg. We need to be disciplined, to study what works scientifically, and – above all - to introduce scientific methods into the practice of entrepreneurship itself.

When we master that, I think we'll know what to do with IPO's again. At least, that's my speculation. In the meantime, it's going to be fun.

Case Study: kaChing, Anatomy of a Pivot

(The following guest post is a new experiment for this blog. It was written by Sarah Milstein[100] in collaboration with kaChing CEO Andy Rachleff. kaChing has been very active in the Lean Startup movement. If you haven't seen it, Pascal's recent presentation on continuous deployment[101] is a must-see; slides are here[102]. In the

[100]http://sarahmilstein.com/

[101]http://vimeo.com/12849143

[102]http://www.slideshare.net/pascallouis/iterate-like-a-whirling-dervish

interests of full disclosure: I am an advisor to kaChing but did not participate in the interviews that led to this case.

With case studies like this, we aim to illustrate specific Lean Startup techniques through the stories of current practitioners. It is written using the information that the company voluntarily shared, and therefore reflects their current thinking and recollections. I am particularly interested in feedback on this case study. Do you find it helpful? Please give us your feedback in the comments. Thanks, Eric)

You probably know that Flickr, the photo-sharing site, started out as an MMOG[103]. And if you're a regular reader of this blog, you may know that IMVU started out as an instant messaging add-on[104]. It's common, perhaps the norm, for startups to pivot[105] like that—to discover that a product is catching on in unintended ways worth pursuing. Yet there's a lot of mystery around pivots, and entrepreneurs ask all the time how you know it's time to commit to a new direction.

To shed some light, I talked with kaChing[106], a destination that enables individual investors to find outstanding money managers to manage their money. The company's audacious goal is to disrupt the $11 Trillion mutual fund industry. The startling part is that kaChing started out as a…Facebook game. That's an epic pivot, like shifting from making solar calculators to powering the Space Shuttle. How'd it happen?

kaChing launched a virtual portfolio management game on

[103]http://www.usatoday.com/tech/products/2006-02-27-flickr_x.htm

[104]http://mixergy.com/ries-lean/

[105]http://www.startuplessonslearned.com/2009/06/pivot-dont-jump-to-new-vision.html

[106]http://kaching.com/

Facebook in January 2008 and a similar version shortly thereafter on kaChing.com. The intent was to discover amateurs who could manage a portfolio as well if not better than professionals (think American Idol) and then facilitate individual investors giving them their real money to manage. In other words, the game would serve as a kind of minor league for the profession. Because kaChing prefers its portfolio managers to have a long track record, the marketplace launch (i.e., the version that would facilitate the investment of real money) was planned for late-2009.

kaChing deployed the game across a slew of platforms, including MySpace, the iPhone, and the Yahoo App Platform. The result? They attracted more than 450,000 portfolios—a decent number for a company that hoped a good percentage would prove out as capable managers. They also hoped a reasonable percentage would realize they were lousy money managers and would then convert to clients.

In the early fall of 2009, as kaChing prepared for its marketplace launch, the management team showed the app—which included real time market data, SEC-grade accounting, analytics, compliance and customer management tools—to a number of investment pros to get feedback and endorsements. One of those pros was John Powers[107], head of the Stanford endowment. He noted the platform would be good not only for amateurs who had proven themselves as outstanding portfolio managers in the game, but also for professional money managers —a group that had insufficient tools for managing and scaling their businesses.

The kaChing system was based on full transparency. A portfolio manager's entire track record & holdings had to be disclosed. The

[107]http://www.stanforddaily.com/tag/john-powers/

company didn't believe professionals would be willing to reveal that level of detail. But Powers's reaction was intriguing enough to prompt Andy Rachleff[108], kaChing's CEO, to call friends who were professional money managers and describe the idea. The response was surprisingly positive.

Andy Mathieson, a founder and managing member at Fairview Capital[109], was particularly supportive. He was unconcerned about transparency, noting the good have nothing to fear. Mathieson signed on to be a money manager in the marketplace launch, committing five years worth of prior transactional data. Mathieson's firm has a minimum investment of $1 million dollars outside of kaChing. On kaChing, consumers could invest in Fairview Capital's strategy with as little as $3k.

When the marketplace launched on October 19, it included seven amateurs who had risen through the game's ranks and four professionals, including Mathieson. Within a month, kaChing observed several interesting things. First, because the amateurs weren't SEC-registered, the site had to refer to them with awkward terms like "geniuses." That was confusing for consumers, who already had to figure out what on kaChing.com was a game and what was real. Second, out of 450,000 gamers, only seven had qualified to become kaChing managers. Third, the company expected hundreds of amateurs who performed poorly in the game to realize they weren't good at investing and therefore become customers. in fact only five people converted into paying customers. Finally, after launch, 30 professional money managers, having read articles on the company, contacted kaChing out of the blue. These managers weren't concerned with trans-

[108]https://www.kaching.com/company/team
[109]http://www.fairviewcap.com/people.html

parency. They were interested in the tools and new distribution medium kaChing provided.

In November, kaChing held an all-hands meeting, circling up chairs in their small Palo Alto office, to discuss whether they should focus solely on professionals and abandon the systems for proving amateurs. “Some people weren’t comfortable because it wasn’t as fun, and one senior engineer thought we’d be losing the part of kaChing that was an enabler for anyone who wanted to make it as a pro,” Rachleff recalls. “But what we really wanted to change was not who manages the money, but who has access to the best possible talent. We’d originally thought we’d need to build a significant business with amateur managers to get professionals to come on board, but fortunately It turns out that wasn’t necessary.”

The staff agreed they could better fulfill their goals by working just with professional managers. In December, they removed the game from kaChing.com. In February, they held another all-hands meeting to talk about shutting down the legacy Facebook game, which still had 60,000 active users. “Everybody felt the burden of supporting all those transactions every day,” says Pascal-Louis Perez, kaChing’s CTO. “It took a ton of our time, and just wasn’t contributing to our long term vision.” That all-hands lasted five minutes.

Which is a nice story. But when kaChing actually shut down each game, hundreds of angry players spewed venom. “We had to ignore them, because they weren’t our target audience – and were never likely to become customers.” says Rachleff.

kaChing says they had the fortitude to make quick decisions and stay the course not just because they’d observed how people were using the marketplace, but also because they’d spoken with hundreds of potential and past customers. To acquire new money managers, the company makes traditional sales calls, which means they've interviewed many, many professionals and gotten a strong sense of their needs. At the same time, whenever a customer closes an account, kaChing contacts the person to find out why; most agree to a short phone interview. (The site has about 700 active paying customers.

Perez says this level of contact, synthesized with their own observations, has given them confidence to make bold decisions. Of the money managers they've interviewed, he notes, "The feedback is consistent; we solve big enough problems for people that we believe they'll come on board."

With 21 employees today, kaChing is devoted to recruiting professional managers and finding product/market fit[110], first for money managers, then for consumers. Thus far the results are encouraging. More than 30 qualified professional money managers have been attracted to the platform and more than $190 million of customer assets have been committed as well.

The kaChing team is quick to note that because they're still closing-in on product/market fit, they're less data-driven than they plan to be once they're in optimizing mode. "We create hypotheses, and test them," says Rachleff. "If something fails, we cut it off. If something seems to succeed, we pursue it aggressively. You have to have the courage of your convictions. With limited data, you have to make tough decisions."

Special thanks to Pascal-Louis Perez for sharing information and making this post possible.

[110]http://www.ashmaurya.com/2010/06/pivot-before-productmarket-fit-optimize-after/

Case Study: SlideShare goes freemium

(Normally, I do not write about companies that are doing a marketing launch. But I have decided to make an exception today, for two reasons. First, SlideShare is a fantastic product (that I use on a regular basis) and an impressive company example of Lean Startup practices in action. Second, their story illustrates a key Lean Startup idea: proving the business in micro-scale. It requires separating the product launch from the marketing launch (see Don't Launch[111]) as well as other staple Lean Startup tactics: minimum viable product, split-testing, customer development and the pivot. This story especially demonstrates that these techniques are not reserved only for tiny startups just starting out. When SlideShare began the journey you're about to read, they already had more than a million visitors a day. Because the stakes were high, they had to successfully use a technique I call Innovation inside the box[112] which is important for entrepreneurs inside established companies of all sizes.

Once again, this case study is a collaboration with Sarah Milstein, who conducted the interviews and wrote the post itself, with some minor edits and commentary from me. As this is a new initiative for this blog, we especially welcome your feedback. Did you find this post useful? One recurring request I hear from Lean Startup practitioners around the world is a desire to see examples of the ideas in action. How are we doing?

In the meantime, take a look at how SlideShare performed a significant pivot while still moving at full speed. -Eric]

[111]http://www.startuplessonslearned.com/2009/03/dont-launch.html

[112]http://www.startuplessonslearned.com/2009/10/innovation-inside-box.html

"The first user experience was actually terrible." Rashmi Sinha, co-founder and CEO of SlideShare[113], describes an early version of the analytics package that's part of the Pro accounts thecompany announced today[114].

If your company is using minimum viable product[115], you've probably said the same thing yourself. A lot. SlideShare, founded in 2007, started experimenting with MVPs and A/B testing this year. Those tools, combined with focused customer interviews, have turbo-charged the company's ability to learn.

What prompted the process change? Early this year, SlideShare launched custom channels[116]. Designed for large businesses, the channels let a company share several types of documents, brand the channel with their own design elements, and then include display advertising, contest promotions, blog aggregation, social media integration and metrics reporting. The idea seemed to SlideShare to be a natural direction. Except it didn't take off. *[I was an early adopter of this feature, and participated in the last marketing launch, as you can see here[117]. Alas, even brilliant marketing adorned with a giant picture of me can't fix the wrong product. -Eric]*

Big companies said they liked the idea, but SlideShare found it hard to close deals. Meanwhile, individuals and smaller companies emailed by the hundreds to say that they wanted the

[113]http://www.slideshare.net/

[114]https://www.slideshare.net/business/premium/plans/1

[115]http://www.blogger.com/%5Bhttp://www.startuplessonslearned.com/2009/08/minimum-viable-product-guide.html

[116]http://blog.slideshare.net/2010/02/03/announcing-branded-channels-on-slideshare/

[117]http://techcrunch.com/2010/05/05/slideshare-now-supports-business-videos-on-professional-content-platform/

features of custom channels, but the sales model—arranged like a media buy—didn't make sense to them.

SlideShare's existing customers had needs that the company's new product—along with its pricing and positioning—simply weren't solving. Realizing it had taken a wrong turn, SlideShare rethought its approach to premium accounts and ultimately performed what we'd call a value capture pivot[118], one where the company changes the way it collects revenue from customers.

The process started with a few moving parts. First, the company began quietly testing subscription pricing plans, initially positing a basic plan and an enterprise version. Second, when an individual or small company signed up, Sinha would email them to ask if they'd be willing to hold a phone interview with her to discuss their experience of the product. Despite the fact that SlideShare's product is well-established with many customers, Sinha still took the critical step of (to use Steve Blank's famous phrase) getting out of the building[119], a particularly important job for founders[120]. Third, SlideShare started holding sales calls with large companies to learn what would prompt them to buy the enterprise version.

"Individuals and small companies wanted analytics, they wanted to know what was happening in social media [for their content], they wanted ad removal and lead gen. Branding was less important to them," says Sinha. Big companies had other needs. "We didn't anticipate at all the control features. For instance, we worked with Pfizer, and they wanted the comments turned off. I

[118]http://www.startuplessonslearned.com/2009/06/pivot-dont-jump-to-new-vision.html

[119]http://steveblank.com/2009/10/08/get-out-of-my-building/

[120]http://steveblank.com/2010/05/13/consultants-don%E2%80%99t-pivot-founders-do/

hadn't thought that would be a feature. But they're regulated, so it makes sense." SlideShare used the two streams of information to segment their market and come up with three plans that recombined the custom channel features in meaningful ways.

But that's just part of the story. As SlideShare was pivoting, it was also trying out two processes to get better results: 1) A/B testing[121] to refine the pricing plans and the page describing them; and 2) MVPs to hone the actual premium features. The combo helped SlideShare learn a lot in short order. *[This is the essential approach to testing a big vision that avoids the "local maximum" trap. See Learning is better than optimization*[122]*. - Eric]*

The company ran landing page splits every two or three days (they initially used Unbounce[123] to generate the pages) and measured them carefully with KISSmetrics[124]. They also used SnapABug[125] for live chat on their site. Between the metrics and the direct customer questions, SlideShare had what Sinha calls "minor learnings and then major shifts."

For example, early iterations of their pricing page included the original, free version of SlideShare. "We realized that was really confusing people," says Sinha. "We don't give you all this Pro plan information right away when you join SlideShare. It's more like, 'If you're already using SlideShare, you might want to try this.'" They removed the core plan, and conversions went up.

---

[121]http://www.startuplessonslearned.com/2008/09/one-line-split-test-or-how-to-ab-all.html

[122]http://www.startuplessonslearned.com/2010/04/learning-is-better-than-optimization.html

[123]http://unbounce.com/

[124]http://www.kissmetrics.com/

[125]http://snapabug.com/

The A/B testing did have its challenges. Because SlideShare has more than a million visitors a day, the team is used to developing features that at least 100,000 people will use. “You get used to having a big impact,” says Sinha. With the split tests, maybe 500 people would see an iteration (SlideShare drove traffic with calls to action around their own site). “You have to get ready to deal with much smaller numbers.”

The MVPs were tricky to implement for emotional reasons, too. Because the SlideShare team was used to giving away a high-value product, engineers balked at charging for a clearly imperfect product. The analytics package, for instance, launched in what Sinha calls “a very crude version; we started off and sold it before we were comfortable with it.”

The saving grace was follow-up interviews. SlideShare asked customers what they had expected in the product; the responses were often literal descriptions. People consistently said they were dying for analytics and specifically that they wanted to track social media and understand the people visiting their content (SlideShare eventually discovered that showing visitors’ locations and timing satisfied people’s needs).

“Charging for something half-baked is really interesting,” says Sinha. “It makes the product team uncomfortable. At the same time, you make sure that you get honest feedback. If the product doesn’t meet customers’ expectations, they cancel. It’s a very honest relationship. On analytics, we got a lot of feedback that it was half-baked, that we sold it under false pretense. But we would just respond honestly and fast and say, ‘Tell us what you want.’ Then we’d get back to them when we had built it.” Customers appreciated the follow-up, and many bought again after the feature had evolved. In this regard, SlideShare used the early adopter feedback not only to improve the product, but too improve its understanding of what subsequent customers would want. [*That is customer development*[126]*. -Eric]*

The marketing launch[127] for SlideShares Pro accounts is today. But the product launch[128] has been happening iteratively over the past months—which means the company is confident in its new offerings. "When we launched custom channels in February, a lot of people reached out and said, "We'd love to buy'," recalls Sinha. "But it never happened." *[Alas, customers don't know*[129] *what they want! -Eric]* Since creating and refining its premium accounts, SlideShare has closed a number of deals, including high-profile accounts like Dell, Cisco and Pfizer.

Sinha notes that Eric Schmidt, in a recent interview[130], said that you find out whether people truly like a product in the second phase after launch. In the first phase, you get a lot of curious people. Only after the buzz has died down do you truly understand what's going on. With careful and continuous learning processes, SlideShare is inverting that idea and going to market with a validated product. That is the essence of proving the business in micro-scale.

[We'll see if the marketing launch results follow the predictions of SlideShare's validated business model. We wish them the best of luck, and hope we can convince them to share their results -

[126]http://www.startuplessonslearned.com/2008/11/what-is-customer-development.html

[127]http://www.startuplessonslearned.com/2009/03/dont-launch.html

[128]http://www.startuplessonslearned.com/2009/03/dont-launch.html

[129]http://www.startuplessonslearned.com/2008/10/when-not-to-listen-to-your-users-when.html

[130]http://www.blogger.com/%5Bhttp://news.cnet.com/8301-13860_3-20012724-56.html

positive or negative - in the near future. In the meantime, good luck and thanks for letting us share your story. -Eric]