Do Things that Don't Scale

July 2013

One of the most common types of advice we give at Y Combinator is to do things that don't scale. A lot of would-be founders believe that startups either take off or don't. You build something, make it available, and if you've made a better mousetrap, people beat a path to your door as promised. Or they don't, in which case the market must not exist. [1]

Actually startups take off because the founders make them take off. There may be a handful that just grew by themselves, but usually it takes some sort of push to get them going. A good metaphor would be the cranks that car engines had before they got electric starters. Once the engine was going, it would keep going, but there was a separate and laborious process to get it going.

Recruit

The most common unscalable thing founders have to do at the start is to recruit users manually. Nearly all startups have to. You can't wait for users to come to you. You have to go out and get them.

Stripe is one of the most successful startups we've funded, and the problem they solved was an urgent one. If anyone could have sat back and waited for users, it was Stripe. But in fact they're famous within YC for aggressive early user acquisition.

Startups building things for other startups have a big pool of potential users in the other companies we've funded, and none took better advantage of it than Stripe. At YC we use the term "Collison installation" for the technique they invented. More diffident founders ask "Will you try our beta?" and if the answer is yes, they say "Great, we'll send you a link." But the Collison brothers weren't going to wait. When anyone agreed to try Stripe they'd say "Right then, give me your laptop" and set them up on the spot.

There are two reasons founders resist going out and recruiting users individually. One is a combination of shyness and laziness. They'd rather sit at home writing code than go out and talk to a bunch of strangers and probably be rejected by most of them. But for a startup to succeed, at least one founder (usually the CEO) will have to spend a lot of time on sales and marketing. [2]

The other reason founders ignore this path is that the absolute numbers seem so small at first. This can't be how the big, famous startups got started, they think. The mistake they make is to underestimate the power of compound growth. We encourage every startup to measure their progress by weekly growth rate. If you have 100 users, you need to get 10 more next week to grow 10% a week. And while 110 may not seem much better than 100, if you keep growing at 10% a week you'll be surprised how big the numbers get. After a year you'll have 14,000 users, and after 2 years you'll have 2 million.

You'll be doing different things when you're acquiring users a thousand at a time, and growth has to slow down eventually. But if the market exists you can usually start by recruiting users manually and then gradually switch to less manual methods. [3]

Airbnb is a classic example of this technique. Marketplaces are so hard to get rolling that you should expect to take heroic measures at first. In Airbnb's case, these consisted of going door to door in New York, recruiting new users and helping existing ones improve their listings. When I remember the Airbnbs during YC, I picture them with rolly bags, because when they showed up for tuesday dinners they'd always just flown back from somewhere.

Fragile

Airbnb now seems like an unstoppable juggernaut, but early on it was so fragile that about 30 days of going out and engaging in person with users made the difference between success and failure.

That initial fragility was not a unique feature of Airbnb. Almost all startups are fragile initially. And that's one of the biggest things inexperienced founders and investors (and reporters and know-it-alls on forums) get wrong about them. They unconsciously judge larval startups by the standards of established ones. They're like someone looking at a newborn baby and concluding "there's no way this tiny creature could ever accomplish anything."

It's harmless if reporters and know-it-alls dismiss your startup. They always get things wrong. It's even ok if investors dismiss your startup; they'll change their minds when they see growth. The big danger is that you'll dismiss your startup yourself. I've seen it happen. I often have to encourage founders who don't see the full potential of what they're building. Even Bill Gates made that mistake. He returned to Harvard for the fall semester after starting Microsoft. He didn't stay long, but he wouldn't have returned at all if he'd realized Microsoft was going to be even a fraction of the size it turned out to be. [4]

The question to ask about an early stage startup is not "is this company taking over the world?" but "how big could this company get if the founders did the right things?" And the right things often seem both laborious and inconsequential at the time. Microsoft can't have seemed very impressive when it was just a couple guys in Albuquerque writing Basic interpreters for a market of a few thousand hobbyists (as they were then called), but in retrospect that was the optimal path to dominating microcomputer software. And I know Brian Chesky and Joe Gebbia didn't feel like they were en route to the big time as they were taking "professional" photos of their first hosts' apartments. They were just trying to survive. But in retrospect that too was the optimal path to dominating a big market.

How do you find users to recruit manually? If you build something to solve your own problems, then you only have to find your peers, which is usually straightforward. Otherwise you'll have to make a more deliberate effort to locate the most promising vein of users. The usual way to do that is to get some initial set of users by doing a comparatively untargeted launch, and then to observe which kind seem most enthusiastic, and seek out more like them. For example, Ben Silbermann noticed that a lot of the earliest Pinterest users were interested in design, so he went to a conference of design bloggers to recruit users, and that worked well. [5]

Delight

You should take extraordinary measures not just to acquire users, but also to make them happy. For as long as they could (which turned out to be surprisingly long), Wufoo sent each new user a hand-written thank you note. Your first users should feel that signing up with you was one of the best choices they ever made. And you in turn should be racking your brains to think of new ways to delight them.

Why do we have to teach startups this? Why is it counterintuitive for founders? Three reasons, I think.

One is that a lot of startup founders are trained as engineers, and customer service is not part of the training of engineers. You're supposed to build things that are robust and elegant, not be slavishly attentive to individual users like some kind of salesperson. Ironically, part of the reason engineering is traditionally averse to handholding is that its traditions date from a time when engineers were less powerful—when they were only in charge of their narrow domain of building things, rather than running the whole show. You can be ornery when you're Scotty, but not when you're Kirk.

Another reason founders don't focus enough on individual customers is that they worry it won't scale. But when founders of larval startups worry about this, I point out that in their current state they have nothing to lose. Maybe if they go out of their way to make existing users super happy, they'll one day have too many to do so much for. That would be a great problem to have. See if you can make it happen. And incidentally, when it does, you'll find that delighting customers scales better than you expected. Partly because you can usually find ways to make anything scale more than you would have predicted, and partly because delighting customers will by then have permeated your culture.

I have never once seen a startup lured down a blind alley by trying too hard to make their initial users happy.

But perhaps the biggest thing preventing founders from realizing how attentive they could be to their users is that they've never experienced such attention themselves. Their standards for customer service have been set by the companies they've been customers of, which are mostly big ones. Tim Cook doesn't send you a hand-written note after you buy a laptop. He can't. But you can. That's one advantage of being small: you can provide a level of service no big company can. [6]

Once you realize that existing conventions are not the upper bound on user experience, it's interesting in a very pleasant way to think about how far you could go to delight your users.

Experience

I was trying to think of a phrase to convey how extreme your attention to users should be, and I realized Steve Jobs had already done it: insanely great. Steve wasn't just using "insanely" as a synonym for "very." He meant it more literally—that one should focus on quality of execution to a degree that in everyday life would be considered pathological.

All the most successful startups we've funded have, and that probably doesn't surprise would-be founders. What novice founders don't get is what insanely great translates to in a larval startup. When Steve Jobs started using that phrase, Apple was already an established company. He meant the Mac (and its documentation and even packaging—such is the nature of obsession) should be insanely well designed and manufactured. That's not hard for engineers to grasp. It's just a more extreme version of designing a robust and elegant product.

What founders have a hard time grasping (and Steve himself might have had a hard time grasping) is what insanely great morphs into as you roll the time slider back to the first couple months of a startup's life. It's not the product that should be insanely great, but the experience of being your user. The product is just one component of that. For a big company it's necessarily the dominant one. But you can and should give users an insanely great experience with an early, incomplete, buggy product, if you make up the difference with attentiveness.

Can, perhaps, but should? Yes. Over-engaging with early users is not just a permissible technique for getting growth rolling. For most successful startups it's a necessary part of the feedback loop that makes the product good. Making a better mousetrap is not an atomic operation. Even if you start the way most successful startups have, by building something you yourself need, the first thing you build is never quite right. And except in domains with big penalties for making mistakes, it's often better not to aim for perfection initially. In software, especially, it usually works best to get something in front of users as soon as it has a quantum of utility, and then see what they do with it. Perfectionism is often an excuse for procrastination, and in any case your initial model of users is always inaccurate, even if you're one of them. [7]

The feedback you get from engaging directly with your earliest users will be the best you ever get. When you're so big you have to resort to focus groups, you'll wish you could go over to your users' homes and offices and watch them use your stuff like you did when there were only a handful of them.

Fire

Sometimes the right unscalable trick is to focus on a deliberately narrow market. It's like keeping a fire contained at first to get it really hot before adding more logs.

That's what Facebook did. At first it was just for Harvard students. In that form it only had a potential market of a few thousand people, but because they felt it was really for them, a critical mass of them signed up. After Facebook stopped being for Harvard students, it remained for students at specific colleges for quite a while. When I interviewed Mark Zuckerberg at Startup School, he said that while it was a lot of work creating course lists for each school, doing that made students feel the site was their natural home.

Any startup that could be described as a marketplace usually has to start in a subset of the market, but this can work for other startups as well. It's always worth asking if there's a subset of the market in which you can get a critical mass of users quickly. [8]

Most startups that use the contained fire strategy do it unconsciously. They build something for themselves and their friends, who happen to be the early adopters, and only realize later that they could offer it to a broader market. The strategy works just as well if you do it unconsciously. The biggest danger of not being consciously aware of this pattern is for those who naively discard part of it. E.g. if you don't build something for yourself and your friends, or even if you do, but you come from the corporate world and your friends are not early adopters, you'll no longer have a perfect initial market handed to you on a platter.

Among companies, the best early adopters are usually other startups. They're more open to new things both by nature and because, having just been started, they haven't made all their choices yet. Plus when they succeed they grow fast, and you with them. It was one of many unforeseen advantages of the YC model (and specifically of making YC big) that B2B startups now have an instant market of hundreds of other startups ready at hand.

Meraki

For hardware startups there's a variant of doing things that don't scale that we call "pulling a Meraki." Although we didn't fund Meraki, the founders were Robert Morris's grad students, so we know their history. They got started by doing something that really doesn't scale: assembling their routers themselves.

Hardware startups face an obstacle that software startups don't. The minimum order for a factory production run is usually several hundred thousand dollars. Which can put you in a catch-22: without a product you can't generate the growth you need to raise the money to manufacture your product. Back when hardware startups had to rely on investors for money, you had to be pretty convincing to overcome this. The arrival of crowdfunding (or more precisely, preorders) has helped a lot. But even so I'd advise startups to pull a Meraki initially if they can. That's what Pebble did. The Pebbles assembled the first several hundred watches themselves. If they hadn't gone through that phase, they probably wouldn't have sold $10 million worth of watches when they did go on Kickstarter.

Like paying excessive attention to early customers, fabricating things yourself turns out to be valuable for hardware startups. You can tweak the design faster when you're the factory, and you learn things you'd never have known otherwise. Eric Migicovsky of Pebble said one of things he learned was "how valuable it was to source good screws." Who knew?

Consult

Sometimes we advise founders of B2B startups to take over-engagement to an extreme, and to pick a single user and act as if they were consultants building something just for that one user. The initial user serves as the form for your mold; keep tweaking till you fit their needs perfectly, and you'll usually find you've made something other users want too. Even if there aren't many of them, there are probably adjacent territories that have more. As long as you can

find just one user who really needs something and can act on that need, you've got a toehold in making something people want, and that's as much as any startup needs initially. [9]

Consulting is the canonical example of work that doesn't scale. But (like other ways of bestowing one's favors liberally) it's safe to do it so long as you're not being paid to. That's where companies cross the line. So long as you're a product company that's merely being extra attentive to a customer, they're very grateful even if you don't solve all their problems. But when they start paying you specifically for that attentiveness—when they start paying you by the hour—they expect you to do everything.

Another consulting-like technique for recruiting initially lukewarm users is to use your software yourselves on their behalf. We did that at Viaweb. When we approached merchants asking if they wanted to use our software to make online stores, some said no, but they'd let us make one for them. Since we would do anything to get users, we did. We felt pretty lame at the time. Instead of organizing big strategic e-commerce partnerships, we were trying to sell luggage and pens and men's shirts. But in retrospect it was exactly the right thing to do, because it taught us how it would feel to merchants to use our software. Sometimes the feedback loop was near instantaneous: in the middle of building some merchant's site I'd find I needed a feature we didn't have, so I'd spend a couple hours implementing it and then resume building the site.

Manual

There's a more extreme variant where you don't just use your software, but are your software. When you only have a small number of users, you can sometimes get away with doing by hand things that you plan to automate later. This lets you launch faster, and when you do finally automate yourself out of the loop, you'll know exactly what to build because you'll have muscle memory from doing it yourself.

When manual components look to the user like software, this technique starts to have aspects of a practical joke. For example, the way Stripe delivered "instant" merchant accounts to its first users was that the founders manually signed them up for traditional merchant accounts behind the scenes.

Some startups could be entirely manual at first. If you can find someone with a problem that needs solving and you can solve it manually, go ahead and do that for as long as you can, and then gradually automate the bottlenecks. It would be a little frightening to be solving users' problems in a way that wasn't yet automatic, but less frightening than the far more common case of having something automatic that doesn't yet solve anyone's problems.

Big

I should mention one sort of initial tactic that usually doesn't work: the Big Launch. I occasionally meet founders who seem to believe startups are projectiles rather than powered aircraft, and that they'll make it big if and only if they're launched with sufficient initial velocity. They want to launch simultaneously in 8 different publications, with embargoes. And on a tuesday, of course, since they read somewhere that's the optimum day to launch something.

It's easy to see how little launches matter. Think of some successful startups. How many of their launches do you remember? All you need from a launch is some initial core of users. How well you're doing a few months later will depend more on how happy you made those users than how many there were of them. [10]

So why do founders think launches matter? A combination of solipsism and laziness. They think what they're building is so great that everyone who hears about it will immediately sign up. Plus it would be so much less work if you could get users merely by broadcasting your existence, rather than recruiting them one at a time. But even if what you're building really is great, getting users will always be a gradual process—partly because great things are usually also novel, but mainly because users have other things to think about.

Partnerships too usually don't work. They don't work for startups in general, but they especially don't work as a way to get growth started. It's a common mistake among inexperienced founders to believe that a partnership with a big company will be their big break. Six months later they're all saying the same thing: that was way more work than we expected, and we ended up getting practically nothing out of it. [11]

It's not enough just to do something extraordinary initially. You have to make an extraordinary effort initially. Any strategy that omits the effort—whether it's expecting a big launch to get you users, or a big partner—is ipso facto suspect.

Vector

The need to do something unscalably laborious to get started is so nearly universal that it might be a good idea to stop thinking of startup ideas as scalars. Instead we should try thinking of them as pairs of what you're going to build, plus the unscalable thing(s) you're going to do initially to get the company going.

It could be interesting to start viewing startup ideas this way, because now that there are two components you can try to be imaginative about the second as well as the first. But in most cases the second component will be what it usually is—recruit users manually and give them an overwhelmingly good experience—and the main benefit of treating startups as vectors will be to remind founders they need to work hard in two dimensions. [12]

In the best case, both components of the vector contribute to your company's DNA: the unscalable things you have to do to get started are not merely a necessary evil, but change the company permanently for the better. If you have to be aggressive about user acquisition when you're small, you'll probably still be aggressive when you're big. If you have to manufacture your own hardware, or use your software on users's behalf, you'll learn things you couldn't have learned otherwise. And most importantly, if you have to work hard to delight users when you only have a handful of them, you'll keep doing it when you have a lot.

How to Start a Startup

Want to start a startup? Get funded by Y Combinator.

March 2005

(This essay is derived from a talk at the Harvard Computer Society.)

You need three things to create a successful startup: to start with good people, to make something customers actually want, and to spend as little money as possible. Most startups that fail do it because they fail at one of these. A startup that does all three will probably succeed.

And that's kind of exciting, when you think about it, because all three are doable. Hard, but doable. And since a startup that succeeds ordinarily makes its founders rich, that implies getting rich is doable too. Hard, but doable.

If there is one message I'd like to get across about startups, that's it. There is no magically difficult step that requires brilliance to solve.

The Idea

In particular, you don't need a brilliant idea to start a startup around. The way a startup makes money is to offer people better technology than they have now. But what people have now is often so bad that it doesn't take brilliance to do better.

Google's plan, for example, was simply to create a search site that didn't suck. They had three new ideas: index more of the Web, use links to rank search results, and have clean, simple web pages with unintrusive keyword-based ads. Above all, they were determined to make a site that was good to use. No doubt there are great technical tricks within Google, but the overall plan was straightforward. And while they probably have bigger ambitions now, this alone brings them a billion dollars a year. [1]

There are plenty of other areas that are just as backward as search was before Google. I can think of several heuristics for generating ideas for startups, but most reduce to this: look at something people are trying to do, and figure out how to do it in a way that doesn't suck.

For example, dating sites currently suck far worse than search did before Google. They all use the same simple-minded model. They seem to have approached the problem by thinking about how to do database matches instead of how dating works in the real world. An undergrad could build something better as a class project. And yet there's a lot of money at stake. Online dating is a valuable business now, and it might be worth a hundred times as much if it worked.

An idea for a startup, however, is only a beginning. A lot of would-be startup founders think the key to the whole process is the initial idea, and from that point all you have to do is execute. Venture capitalists know better. If you go to VC firms with a brilliant idea that you'll tell them about if they sign a nondisclosure agreement, most will tell you to get lost. That shows how much a mere idea is worth. The market price is less than the inconvenience of signing an NDA.

Another sign of how little the initial idea is worth is the number of startups that change their plan en route. Microsoft's original plan was to make money selling programming languages, of all things. Their current business model didn't occur to them until IBM dropped it in their lap five years later.

Ideas for startups are worth something, certainly, but the trouble is, they're not transferrable. They're not something you could hand to someone else to execute. Their value is mainly as starting points: as questions for the people who had them to continue thinking about.

What matters is not ideas, but the people who have them. Good people can fix bad ideas, but good ideas can't save bad people.

People

What do I mean by good people? One of the best tricks I learned during our startup was a rule for deciding who to hire. Could you describe the person as an animal? It might be hard to translate that into another language, but I think everyone in the US knows what it means. It means someone who takes their work a little too seriously; someone who does what they do so well that they pass right through professional and cross over into obsessive.

What it means specifically depends on the job: a salesperson who just won't take no for an answer; a hacker who will stay up till 4:00 AM rather than go to bed leaving code with a bug in it; a PR person who will cold-call New York Times reporters on their cell phones; a graphic designer who feels physical pain when something is two millimeters out of place.

Almost everyone who worked for us was an animal at what they did. The woman in charge of sales was so tenacious that I used to feel sorry for potential customers on the phone with her. You could sense them squirming on the hook, but you knew there would be no rest for them till they'd signed up.

If you think about people you know, you'll find the animal test is easy to apply. Call the person's image to mind and imagine the sentence "so-and-so is an animal." If you laugh, they're not. You don't need or perhaps even want this quality in big companies, but you need it in a startup.

For programmers we had three additional tests. Was the person genuinely smart? If so, could they actually get things done? And finally, since a few good hackers have unbearable personalities, could we stand to have them around?

That last test filters out surprisingly few people. We could bear any amount of nerdiness if someone was truly smart. What we couldn't stand were people with a lot of attitude. But most of those weren't truly smart, so our third test was largely a restatement of the first.

When nerds are unbearable it's usually because they're trying too hard to seem smart. But the smarter they are, the less pressure they feel to act smart. So as a rule you can recognize genuinely smart people by their ability to say things like "I don't know," "Maybe you're right," and "I don't understand x well enough."

This technique doesn't always work, because people can be influenced by their environment. In the MIT CS department, there seems to be a tradition of acting like a brusque know-it-all. I'm told it derives ultimately from Marvin Minsky, in the same way the classic airline pilot manner is said to derive from Chuck Yeager. Even genuinely smart people start to act this way there, so you have to make allowances.

It helped us to have Robert Morris, who is one of the readiest to say "I don't know" of anyone I've met. (At least, he was before he became a professor at MIT.) No one dared put on attitude around Robert, because he was obviously smarter than they were and yet had zero attitude himself.

Like most startups, ours began with a group of friends, and it was through personal contacts that we got most of the people we hired. This is a crucial difference between startups and big companies. Being friends with someone for even a couple days will tell you more than companies could ever learn in interviews. [2]

It's no coincidence that startups start around universities, because that's where smart people meet. It's not what people learn in classes at MIT and Stanford that has made technology companies spring up around them. They could sing campfire songs in the classes so long as admissions worked the same.

If you start a startup, there's a good chance it will be with people you know from college or grad school. So in theory you ought to try to make friends with as many smart people as you can in school, right? Well, no. Don't make a conscious effort to schmooze; that doesn't work well with hackers.

What you should do in college is work on your own projects. Hackers should do this even if they don't plan to start startups, because it's the only real way to learn how to program. In some cases you may collaborate with other students, and this is the best way to get to know good hackers. The project may even grow into a startup. But once again, I wouldn't aim too directly at either target. Don't force things; just work on stuff you like with people you like.

Ideally you want between two and four founders. It would be hard to start with just one. One person would find the moral weight of starting a company hard to bear. Even Bill Gates, who seems to be able to bear a good deal of moral weight, had to have a co-founder. But you don't want so many founders that the company starts to look like a group photo. Partly because you don't need a lot of people at first, but mainly because the more founders you have, the worse disagreements you'll have. When there are just two or three founders, you know you have to resolve disputes immediately or perish. If there are seven or eight, disagreements can linger and harden into factions. You don't want mere voting; you need unanimity.

In a technology startup, which most startups are, the founders should include technical people. During the Internet Bubble there were a number of startups founded by business people who then went looking for hackers to create their product for them. This doesn't work well. Business people are bad at deciding what to do with technology, because they don't know what the options are, or which kinds of problems are hard and which are easy. And when business people try to hire hackers, they can't tell which ones are good. Even other hackers have a hard time doing that. For business people it's roulette.

Do the founders of a startup have to include business people? That depends. We thought so when we started ours, and we asked several people who were said to know about this mysterious thing called "business" if they would be the president. But they all said no, so I had to do it myself. And what I discovered was that business was no great mystery. It's not something like physics or medicine that requires extensive study. You just try to get people to pay you for stuff.

I think the reason I made such a mystery of business was that I was disgusted by the idea of doing it. I wanted to work in the pure, intellectual world of software, not deal with customers' mundane problems. People who don't want to get dragged into some kind of work often develop a protective incompetence at it. Paul Erdos was particularly good at this. By seeming unable even to cut a grapefruit in half (let alone go to the store and buy one), he forced other people to do such things for him, leaving all his time free for math. Erdos was an extreme case, but most husbands use the same trick to some degree.

Once I was forced to discard my protective incompetence, I found that business was neither so hard nor so boring as I feared. There are esoteric areas of business that are quite hard, like tax law or the pricing of derivatives, but you don't need to know about those in a startup. All you need to know about business to run a startup are commonsense things people knew before there were business schools, or even universities.

If you work your way down the Forbes 400 making an x next to the name of each person with an MBA, you'll learn something important about business school. After Warren Buffett, you don't hit another MBA till number 22, Phil Knight, the CEO of Nike. There are only 5 MBAs in the top 50. What you notice in the Forbes 400 are a lot of people with technical backgrounds. Bill Gates, Steve Jobs, Larry Ellison, Michael Dell, Jeff Bezos, Gordon Moore. The rulers of the technology business tend to come from technology, not business. So if you want to invest two years in something that will help you succeed in business, the evidence suggests you'd do better to learn how to hack than get an MBA. [3]

There is one reason you might want to include business people in a startup, though: because you have to have at least one person willing and able to focus on what customers want. Some believe only business people can do this--that hackers can implement software, but not design it. That's nonsense. There's nothing about knowing how to program that prevents hackers from understanding users, or about not knowing how to program that magically enables business people to understand them.

If you can't understand users, however, you should either learn how or find a co-founder who can. That is the single most important issue for technology startups, and the rock that sinks more of them than anything else.

What Customers Want

It's not just startups that have to worry about this. I think most businesses that fail do it because they don't give customers what they want. Look at restaurants. A large percentage fail, about a quarter in the first year. But can you think of one restaurant that had really good food and went out of business?

Restaurants with great food seem to prosper no matter what. A restaurant with great food can be expensive, crowded, noisy, dingy, out of the way, and even have bad service, and people will keep coming. It's true that a restaurant with mediocre food can sometimes attract customers through gimmicks. But that approach is very risky. It's more straightforward just to make the food good.

It's the same with technology. You hear all kinds of reasons why startups fail. But can you think of one that had a massively popular product and still failed?

In nearly every failed startup, the real problem was that customers didn't want the product. For most, the cause of death is listed as "ran out of funding," but that's only the immediate cause. Why couldn't they get more funding? Probably because the product was a dog, or never seemed likely to be done, or both.

When I was trying to think of the things every startup needed to do, I almost included a fourth: get a version 1 out as soon as you can. But I decided not to, because that's implicit in making something customers want. The only way to make something customers want is to get a prototype in front of them and refine it based on their reactions.

The other approach is what I call the "Hail Mary" strategy. You make elaborate plans for a product, hire a team of engineers to develop it (people who do this tend to use the term "engineer" for hackers), and then find after a year that you've spent two million dollars to develop something no one wants. This was not uncommon during the Bubble, especially in companies run by business types, who thought of software development as something terrifying that therefore had to be carefully planned.

We never even considered that approach. As a Lisp hacker, I come from the tradition of rapid prototyping. I would not claim (at least, not here) that this is the right way to write every program, but it's certainly the right way to write

software for a startup. In a startup, your initial plans are almost certain to be wrong in some way, and your first priority should be to figure out where. The only way to do that is to try implementing them.

Like most startups, we changed our plan on the fly. At first we expected our customers to be Web consultants. But it turned out they didn't like us, because our software was easy to use and we hosted the site. It would be too easy for clients to fire them. We also thought we'd be able to sign up a lot of catalog companies, because selling online was a natural extension of their existing business. But in 1996 that was a hard sell. The middle managers we talked to at catalog companies saw the Web not as an opportunity, but as something that meant more work for them.

We did get a few of the more adventurous catalog companies. Among them was Frederick's of Hollywood, which gave us valuable experience dealing with heavy loads on our servers. But most of our users were small, individual merchants who saw the Web as an opportunity to build a business. Some had retail stores, but many only existed online. And so we changed direction to focus on these users. Instead of concentrating on the features Web consultants and catalog companies would want, we worked to make the software easy to use.

I learned something valuable from that. It's worth trying very, very hard to make technology easy to use. Hackers are so used to computers that they have no idea how horrifying software seems to normal people. Stephen Hawking's editor told him that every equation he included in his book would cut sales in half. When you work on making technology easier to use, you're riding that curve up instead of down. A 10% improvement in ease of use doesn't just increase your sales 10%. It's more likely to double your sales.

How do you figure out what customers want? Watch them. One of the best places to do this was at trade shows. Trade shows didn't pay as a way of getting new customers, but they were worth it as market research. We didn't just give canned presentations at trade shows. We used to show people how to build real, working stores. Which meant we got to watch as they used our software, and talk to them about what they needed.

No matter what kind of startup you start, it will probably be a stretch for you, the founders, to understand what users want. The only kind of software you can build without studying users is the sort for which you are the typical user. But this is just the kind that tends to be open source: operating systems, programming languages, editors, and so on. So if you're developing technology for money, you're probably not going to be developing it for people like you. Indeed, you can use this as a way to generate ideas for startups: what do people who are not like you want from technology?

When most people think of startups, they think of companies like Apple or Google. Everyone knows these, because they're big consumer brands. But for every startup like that, there are twenty more that operate in niche markets or live quietly down in the infrastructure. So if you start a successful startup, odds are you'll start one of those.

Another way to say that is, if you try to start the kind of startup that has to be a big consumer brand, the odds against succeeding are steeper. The best odds are in niche markets. Since startups make money by offering people something better than they had before, the best opportunities are where things suck most. And it would be hard to find a place where things suck more than in corporate IT departments. You would not believe the amount of money companies spend on software, and the crap they get in return. This imbalance equals opportunity.

If you want ideas for startups, one of the most valuable things you could do is find a middle-sized non-technology company and spend a couple weeks just watching what they do with computers. Most good hackers have no more idea of the horrors perpetrated in these places than rich Americans do of what goes on in Brazilian slums.

Start by writing software for smaller companies, because it's easier to sell to them. It's worth so much to sell stuff to big companies that the people selling them the crap they currently use spend a lot of time and money to do it. And while you can outhack Oracle with one frontal lobe tied behind your back, you can't outsell an Oracle salesman. So if you want to win through better technology, aim at smaller customers. [4]

They're the more strategically valuable part of the market anyway. In technology, the low end always eats the high end. It's easier to make an inexpensive product more powerful than to make a powerful product cheaper. So the products that start as cheap, simple options tend to gradually grow more powerful till, like water rising in a room, they squash the "high-end" products against the ceiling. Sun did this to mainframes, and Intel is doing it to Sun. Microsoft Word did it to desktop publishing software like Interleaf and Framemaker. Mass-market digital cameras are doing it to the expensive models made for professionals. Avid did it to the manufacturers of specialized video editing systems, and now Apple is doing it to Avid. Henry Ford did it to the car makers that preceded him. If you build the simple, inexpensive option, you'll not only find it easier to sell at first, but you'll also be in the best position to conquer the rest of the market.

It's very dangerous to let anyone fly under you. If you have the cheapest, easiest product, you'll own the low end. And if you don't, you're in the crosshairs of whoever does.

Raising Money

To make all this happen, you're going to need money. Some startups have been self-funding-- Microsoft for example-- but most aren't. I think it's wise to take money from investors. To be self-funding, you have to start as a consulting company, and it's hard to switch from that to a product company.

Financially, a startup is like a pass/fail course. The way to get rich from a startup is to maximize the company's chances of succeeding, not to maximize the amount of stock you retain. So if you can trade stock for something that improves your odds, it's probably a smart move.

To most hackers, getting investors seems like a terrifying and mysterious process. Actually it's merely tedious. I'll try to give an outline of how it works.

The first thing you'll need is a few tens of thousands of dollars to pay your expenses while you develop a prototype. This is called seed capital. Because so little money is involved, raising seed capital is comparatively easy-- at least in the sense of getting a quick yes or no.

Usually you get seed money from individual rich people called "angels." Often they're people who themselves got rich from technology. At the seed stage, investors don't expect you to have an elaborate business plan. Most know that they're supposed to decide quickly. It's not unusual to get a check within a week based on a half-page agreement.

We started Viaweb with $10,000 of seed money from our friend Julian. But he gave us a lot more than money. He's a former CEO and also a corporate lawyer, so he gave us a lot of valuable advice about business, and also did all the legal work of getting us set up as a company. Plus he introduced us to one of the two angel investors who supplied our next round of funding.

Some angels, especially those with technology backgrounds, may be satisfied with a demo and a verbal description of what you plan to do. But many will want a copy of your business plan, if only to remind themselves what they invested in.

Our angels asked for one, and looking back, I'm amazed how much worry it caused me. "Business plan" has that word "business" in it, so I figured it had to be something I'd have to read a book about business plans to write. Well, it doesn't. At this stage, all most investors expect is a brief description of what you plan to do and how you're going to make money from it, and the resumes of the founders. If you just sit down and write out what you've been saying to one another, that should be fine. It shouldn't take more than a couple hours, and you'll probably find that writing it all down gives you more ideas about what to do.

For the angel to have someone to make the check out to, you're going to have to have some kind of company. Merely incorporating yourselves isn't hard. The problem is, for the company to exist, you have to decide who the founders are, and how much stock they each have. If there are two founders with the same qualifications who are both equally committed to the business, that's easy. But if you have a number of people who are expected to contribute in varying degrees, arranging the proportions of stock can be hard. And once you've done it, it tends to be set in stone.

I have no tricks for dealing with this problem. All I can say is, try hard to do it right. I do have a rule of thumb for recognizing when you have, though. When everyone feels they're getting a slightly bad deal, that they're doing more than they should for the amount of stock they have, the stock is optimally apportioned.

There is more to setting up a company than incorporating it, of course: insurance, business license, unemployment compensation, various things with the IRS. I'm not even sure what the list is, because we, ah, skipped all that. When we got real funding near the end of 1996, we hired a great CFO, who fixed everything retroactively. It turns out that no one comes and arrests you if you don't do everything you're supposed to when starting a company. And a good thing too, or a lot of startups would never get started. [5]

It can be dangerous to delay turning yourself into a company, because one or more of the founders might decide to split off and start another company doing the same thing. This does happen. So when you set up the company, as well as as apportioning the stock, you should get all the founders to sign something agreeing that everyone's ideas belong to this company, and that this company is going to be everyone's only job.

[If this were a movie, ominous music would begin here.]

While you're at it, you should ask what else they've signed. One of the worst things that can happen to a startup is to run into intellectual property problems. We did, and it came closer to killing us than any competitor ever did.

As we were in the middle of getting bought, we discovered that one of our people had, early on, been bound by an agreement that said all his ideas belonged to the giant company that was paying for him to go to grad school. In theory, that could have meant someone else owned big chunks of our software. So the acquisition came to a screeching halt while we tried to sort this out. The problem was, since we'd been about to be acquired, we'd allowed ourselves to run low on cash. Now we needed to raise more to keep going. But it's hard to raise money with an IP cloud over your head, because investors can't judge how serious it is.

Our existing investors, knowing that we needed money and had nowhere else to get it, at this point attempted certain gambits which I will not describe in detail, except to remind readers that the word "angel" is a metaphor. The founders thereupon proposed to walk away from the company, after giving the investors a brief tutorial on how to administer the servers themselves. And while this was happening, the acquirers used the delay as an excuse to welch on the deal.

Miraculously it all turned out ok. The investors backed down; we did another round of funding at a reasonable valuation; the giant company finally gave us a piece of paper saying they didn't own our software; and six months later we were bought by Yahoo for much more than the earlier acquirer had agreed to pay. So we were happy in the end, though the experience probably took several years off my life.

Don't do what we did. Before you consummate a startup, ask everyone about their previous IP history.

Once you've got a company set up, it may seem presumptuous to go knocking on the doors of rich people and asking them to invest tens of thousands of dollars in something that is really just a bunch of guys with some ideas. But when you look at it from the rich people's point of view, the picture is more encouraging. Most rich people are looking for good investments. If you really think you have a chance of succeeding, you're doing them a favor by letting them invest. Mixed with any annoyance they might feel about being approached will be the thought: are these guys the next Google?

Usually angels are financially equivalent to founders. They get the same kind of stock and get diluted the same amount in future rounds. How much stock should they get? That depends on how ambitious you feel. When you offer x percent of your company for y dollars, you're implicitly claiming a certain value for the whole company. Venture investments are usually described in terms of that number. If you give an investor new shares equal to 5% of those already outstanding in return for $100,000, then you've done the deal at a pre-money valuation of $2 million.

How do you decide what the value of the company should be? There is no rational way. At this stage the company is just a bet. I didn't realize that when we were raising money. Julian thought we ought to value the company at several million dollars. I thought it was preposterous to claim that a couple thousand lines of code, which was all we had at the time, were worth several million dollars. Eventually we settled on one millon, because Julian said no one would invest in a company with a valuation any lower. [6]

What I didn't grasp at the time was that the valuation wasn't just the value of the code we'd written so far. It was also the value of our ideas, which turned out to be right, and of all the future work we'd do, which turned out to be a lot.

The next round of funding is the one in which you might deal with actual venture capital firms. But don't wait till you've burned through your last round of funding to start approaching them. VCs are slow to make up their minds. They can take months. You don't want to be running out of money while you're trying to negotiate with them.

Getting money from an actual VC firm is a bigger deal than getting money from angels. The amounts of money involved are larger, millions usually. So the deals take longer, dilute you more, and impose more onerous conditions.

Sometimes the VCs want to install a new CEO of their own choosing. Usually the claim is that you need someone mature and experienced, with a business background. Maybe in some cases this is true. And yet Bill Gates was young and inexperienced and had no business background, and he seems to have done ok. Steve Jobs got booted out of his own company by someone mature and experienced, with a business background, who then proceeded to ruin the company. So I think people who are mature and experienced, with a business background, may be overrated. We used to call these guys "newscasters," because they had neat hair and spoke in deep, confident voices, and generally didn't know much more than they read on the teleprompter.

We talked to a number of VCs, but eventually we ended up financing our startup entirely with angel money. The main reason was that we feared a brand-name VC firm would stick us with a newscaster as part of the deal. That might have been ok if he was content to limit himself to talking to the press, but what if he wanted to have a say in running the company? That would have led to disaster, because our software was so complex. We were a company whose whole m.o. was to win through better technology. The strategic decisions were mostly decisions about technology, and we didn't need any help with those.

This was also one reason we didn't go public. Back in 1998 our CFO tried to talk me into it. In those days you could go public as a dogfood portal, so as a company with a real product and real revenues, we might have done well. But I feared it would have meant taking on a newscaster-- someone who, as they say, "can talk Wall Street's language."

I'm happy to see Google is bucking that trend. They didn't talk Wall Street's language when they did their IPO, and Wall Street didn't buy. And now Wall Street is collectively kicking itself. They'll pay attention next time. Wall Street learns new languages fast when money is involved.

You have more leverage negotiating with VCs than you realize. The reason is other VCs. I know a number of VCs now, and when you talk to them you realize that it's a seller's market. Even now there is too much money chasing too few good deals.

VCs form a pyramid. At the top are famous ones like Sequoia and Kleiner Perkins, but beneath those are a huge number you've never heard of. What they all have in common is that a dollar from them is worth one dollar. Most VCs will tell you that they don't just provide money, but connections and advice. If you're talking to Vinod Khosla or John Doerr or Mike Moritz, this is true. But such advice and connections can come very expensive. And as you go down the food chain the VCs get rapidly dumber. A few steps down from the top you're basically talking to bankers who've picked up a few new vocabulary words from reading Wired. (Does your product use XML?) So I'd advise you to be skeptical about claims of experience and connections. Basically, a VC is a source of money. I'd be inclined to go with whoever offered the most money the soonest with the least strings attached.

You may wonder how much to tell VCs. And you should, because some of them may one day be funding your competitors. I think the best plan is not to be overtly secretive, but not to tell them everything either. After all, as most VCs say, they're more interested in the people than the ideas. The main reason they want to talk about your idea is to judge you, not the idea. So as long as you seem like you know what you're doing, you can probably keep a few things back from them. [7]

Talk to as many VCs as you can, even if you don't want their money, because a) they may be on the board of someone who will buy you, and b) if you seem impressive, they'll be discouraged from investing in your competitors. The most efficient way to reach VCs, especially if you only want them to know about you and don't want their money, is at the conferences that are occasionally organized for startups to present to them.

Not Spending It

When and if you get an infusion of real money from investors, what should you do with it? Not spend it, that's what. In nearly every startup that fails, the proximate cause is running out of money. Usually there is something deeper wrong. But even a proximate cause of death is worth trying hard to avoid.

During the Bubble many startups tried to "get big fast." Ideally this meant getting a lot of customers fast. But it was easy for the meaning to slide over into hiring a lot of people fast.

Of the two versions, the one where you get a lot of customers fast is of course preferable. But even that may be overrated. The idea is to get there first and get all the users, leaving none for competitors. But I think in most businesses the advantages of being first to market are not so overwhelmingly great. Google is again a case in point. When they appeared it seemed as if search was a mature market, dominated by big players who'd spent millions to build their brands: Yahoo, Lycos, Excite, Infoseek, Altavista, Inktomi. Surely 1998 was a little late to arrive at the party.

But as the founders of Google knew, brand is worth next to nothing in the search business. You can come along at any point and make something better, and users will gradually seep over to you. As if to emphasize the point, Google never did any advertising. They're like dealers; they sell the stuff, but they know better than to use it themselves.

The competitors Google buried would have done better to spend those millions improving their software. Future startups should learn from that mistake. Unless you're in a market where products are as undifferentiated as cigarettes or vodka or laundry detergent, spending a lot on brand advertising is a sign of breakage. And few if any Web businesses are so undifferentiated. The dating sites are running big ad campaigns right now, which is all the more evidence they're ripe for the picking. (Fee, fie, fo, fum, I smell a company run by marketing guys.)

We were compelled by circumstances to grow slowly, and in retrospect it was a good thing. The founders all learned to do every job in the company. As well as writing software, I had to do sales and customer support. At sales I was not very good. I was persistent, but I didn't have the smoothness of a good salesman. My message to potential customers was: you'd be stupid not to sell online, and if you sell online you'd be stupid to use anyone else's software. Both statements were true, but that's not the way to convince people.

I was great at customer support though. Imagine talking to a customer support person who not only knew everything about the product, but would apologize abjectly if there was a bug, and then fix it immediately, while you were on the phone with them. Customers loved us. And we loved them, because when you're growing slow by word of mouth, your first batch of users are the ones who were smart enough to find you by themselves. There is nothing more valuable, in the early stages of a startup, than smart users. If you listen to them, they'll tell you exactly how to make a winning product. And not only will they give you this advice for free, they'll pay you.

We officially launched in early 1996. By the end of that year we had about 70 users. Since this was the era of "get big fast," I worried about how small and obscure we were. But in fact we were doing exactly the right thing. Once you get big (in users or employees) it gets hard to change your product. That year was effectively a laboratory for improving our software. By the end of it, we were so far ahead of our competitors that they never had a hope of catching up. And since all the hackers had spent many hours talking to users, we understood online commerce way better than anyone else.

That's the key to success as a startup. There is nothing more important than understanding your business. You might think that anyone in a business must, ex officio, understand it. Far from it. Google's secret weapon was simply that they understood search. I was working for Yahoo when Google appeared, and Yahoo didn't understand search. I know because I once tried to convince the powers that be that we had to make search better, and I got in reply what was then the party line about it: that Yahoo was no longer a mere "search engine." Search was now only a small percentage of our page views, less than one month's growth, and now that we were established as a "media company," or "portal," or whatever we were, search could safely be allowed to wither and drop off, like an umbilical cord.

Well, a small fraction of page views they may be, but they are an important fraction, because they are the page views that Web sessions start with. I think Yahoo gets that now.

Google understands a few other things most Web companies still don't. The most important is that you should put users before advertisers, even though the advertisers are paying and users aren't. One of my favorite bumper stickers reads "if the people lead, the leaders will follow." Paraphrased for the Web, this becomes "get all the users, and the advertisers will follow." More generally, design your product to please users first, and then think about how to make money from it. If you don't put users first, you leave a gap for competitors who do.

To make something users love, you have to understand them. And the bigger you are, the harder that is. So I say "get big slow." The slower you burn through your funding, the more time you have to learn.

The other reason to spend money slowly is to encourage a culture of cheapness. That's something Yahoo did understand. David Filo's title was "Chief Yahoo," but he was proud that his unofficial title was "Cheap Yahoo." Soon after we arrived at Yahoo, we got an email from Filo, who had been crawling around our directory hierarchy, asking if it was really necessary to store so much of our data on expensive RAID drives. I was impressed by that. Yahoo's market cap then was already in the billions, and they were still worrying about wasting a few gigs of disk space.

When you get a couple million dollars from a VC firm, you tend to feel rich. It's important to realize you're not. A rich company is one with large revenues. This money isn't revenue. It's money investors have given you in the hope you'll be able to generate revenues. So despite those millions in the bank, you're still poor.

For most startups the model should be grad student, not law firm. Aim for cool and cheap, not expensive and impressive. For us the test of whether a startup understood this was whether they had Aeron chairs. The Aeron came out during the Bubble and was very popular with startups. Especially the type, all too common then, that was like a bunch of kids playing house with money supplied by VCs. We had office chairs so cheap that the arms all fell off. This was slightly embarrassing at the time, but in retrospect the grad-studenty atmosphere of our office was another of those things we did right without knowing it.

Our offices were in a wooden triple-decker in Harvard Square. It had been an apartment until about the 1970s, and there was still a claw-footed bathtub in the bathroom. It must once have been inhabited by someone fairly eccentric, because a lot of the chinks in the walls were stuffed with aluminum foil, as if to protect against cosmic rays. When eminent visitors came to see us, we were a bit sheepish about the low production values. But in fact that place was the perfect space for a startup. We felt like our role was to be impudent underdogs instead of corporate stuffed shirts, and that is exactly the spirit you want.

An apartment is also the right kind of place for developing software. Cube farms suck for that, as you've probably discovered if you've tried it. Ever notice how much easier it is to hack at home than at work? So why not make work more like home?

When you're looking for space for a startup, don't feel that it has to look professional. Professional means doing good work, not elevators and glass walls. I'd advise most startups to avoid corporate space at first and just rent an apartment. You want to live at the office in a startup, so why not have a place designed to be lived in as your office?

Besides being cheaper and better to work in, apartments tend to be in better locations than office buildings. And for a startup location is very important. The key to productivity is for people to come back to work after dinner. Those hours after the phone stops ringing are by far the best for getting work done. Great things happen when a group of employees go out to dinner together, talk over ideas, and then come back to their offices to implement them. So you want to be in a place where there are a lot of restaurants around, not some dreary office park that's a wasteland after 6:00 PM. Once a company shifts over into the model where everyone drives home to the suburbs for dinner, however late, you've lost something extraordinarily valuable. God help you if you actually start in that mode.

If I were going to start a startup today, there are only three places I'd consider doing it: on the Red Line near Central, Harvard, or Davis Squares (Kendall is too sterile); in Palo Alto on University or California Aves; and in Berkeley immediately north or south of campus. These are the only places I know that have the right kind of vibe.

The most important way to not spend money is by not hiring people. I may be an extremist, but I think hiring people is the worst thing a company can do. To start with, people are a recurring expense, which is the worst kind. They also tend to cause you to grow out of your space, and perhaps even move to the sort of uncool office building that will make your software worse. But worst of all, they slow you down: instead of sticking your head in someone's office and checking out an idea with them, eight people have to have a meeting about it. So the fewer people you can hire, the better.

During the Bubble a lot of startups had the opposite policy. They wanted to get "staffed up" as soon as possible, as if you couldn't get anything done unless there was someone with the corresponding job title. That's big company thinking. Don't hire people to fill the gaps in some a priori org chart. The only reason to hire someone is to do something you'd like to do but can't.

If hiring unnecessary people is expensive and slows you down, why do nearly all companies do it? I think the main reason is that people like the idea of having a lot of people working for them. This weakness often extends right up to the CEO. If you ever end up running a company, you'll find the most common question people ask is how many employees you have. This is their way of weighing you. It's not just random people who ask this; even reporters do. And they're going to be a lot more impressed if the answer is a thousand than if it's ten.

This is ridiculous, really. If two companies have the same revenues, it's the one with fewer employees that's more impressive. When people used to ask me how many people our startup had, and I answered "twenty," I could see them thinking that we didn't count for much. I used to want to add "but our main competitor, whose ass we regularly kick, has a hundred and forty, so can we have credit for the larger of the two numbers?"

As with office space, the number of your employees is a choice between seeming impressive, and being impressive. Any of you who were nerds in high school know about this choice. Keep doing it when you start a company.

Should You?

But should you start a company? Are you the right sort of person to do it? If you are, is it worth it?

More people are the right sort of person to start a startup than realize it. That's the main reason I wrote this. There could be ten times more startups than there are, and that would probably be a good thing.

I was, I now realize, exactly the right sort of person to start a startup. But the idea terrified me at first. I was forced into it because I was a Lisp hacker. The company I'd been consulting for seemed to be running into trouble, and there were not a lot of other companies using Lisp. Since I couldn't bear the thought of programming in another language (this was 1995, remember, when "another language" meant C++) the only option seemed to be to start a new company using Lisp.

I realize this sounds far-fetched, but if you're a Lisp hacker you'll know what I mean. And if the idea of starting a startup frightened me so much that I only did it out of necessity, there must be a lot of people who would be good at it but who are too intimidated to try.

So who should start a startup? Someone who is a good hacker, between about 23 and 38, and who wants to solve the money problem in one shot instead of getting paid gradually over a conventional working life.

I can't say precisely what a good hacker is. At a first rate university this might include the top half of computer science majors. Though of course you don't have to be a CS major to be a hacker; I was a philosophy major in college.

It's hard to tell whether you're a good hacker, especially when you're young. Fortunately the process of starting startups tends to select them automatically. What drives people to start startups is (or should be) looking at existing technology and thinking, don't these guys realize they should be doing x, y, and z? And that's also a sign that one is a good hacker.

I put the lower bound at 23 not because there's something that doesn't happen to your brain till then, but because you need to see what it's like in an existing business before you try running your own. The business doesn't have to be a startup. I spent a year working for a software company to pay off my college loans. It was the worst year of my adult life, but I learned, without realizing it at the time, a lot of valuable lessons about the software business. In this case they were mostly negative lessons: don't have a lot of meetings; don't have chunks of code that multiple people own; don't have a sales guy running the company; don't make a high-end product; don't let your code get too big; don't leave finding bugs to QA people; don't go too long between releases; don't isolate developers from users; don't move from Cambridge to Route 128; and so on. [8] But negative lessons are just as valuable as positive ones. Perhaps even more valuable: it's hard to repeat a brilliant performance, but it's straightforward to avoid errors. [9]

The other reason it's hard to start a company before 23 is that people won't take you seriously. VCs won't trust you, and will try to reduce you to a mascot as a condition of funding. Customers will worry you're going to flake out and leave them stranded. Even you yourself, unless you're very unusual, will feel your age to some degree; you'll find it awkward to be the boss of someone much older than you, and if you're 21, hiring only people younger rather limits your options.

Some people could probably start a company at 18 if they wanted to. Bill Gates was 19 when he and Paul Allen started Microsoft. (Paul Allen was 22, though, and that probably made a difference.) So if you're thinking, I don't care what he says, I'm going to start a company now, you may be the sort of person who could get away with it.

The other cutoff, 38, has a lot more play in it. One reason I put it there is that I don't think many people have the physical stamina much past that age. I used to work till 2:00 or 3:00 AM every night, seven days a week. I don't know if I could do that now.

Also, startups are a big risk financially. If you try something that blows up and leaves you broke at 26, big deal; a lot of 26 year olds are broke. By 38 you can't take so many risks-- especially if you have kids.

My final test may be the most restrictive. Do you actually want to start a startup? What it amounts to, economically, is compressing your working life into the smallest possible space. Instead of working at an ordinary rate for 40 years, you work like hell for four. And maybe end up with nothing-- though in that case it probably won't take four years.

During this time you'll do little but work, because when you're not working, your competitors will be. My only leisure activities were running, which I needed to do to keep working anyway, and about fifteen minutes of reading a night. I had a girlfriend for a total of two months during that three year period. Every couple weeks I would take a few hours off to visit a used bookshop or go to a friend's house for dinner. I went to visit my family twice. Otherwise I just worked.

Working was often fun, because the people I worked with were some of my best friends. Sometimes it was even technically interesting. But only about 10% of the time. The best I can say for the other 90% is that some of it is funnier in hindsight than it seemed then. Like the time the power went off in Cambridge for about six hours, and we made the mistake of trying to start a gasoline powered generator inside our offices. I won't try that again.

I don't think the amount of bullshit you have to deal with in a startup is more than you'd endure in an ordinary working life. It's probably less, in fact; it just seems like a lot because it's compressed into a short period. So mainly what a startup buys you is time. That's the way to think about it if you're trying to decide whether to start one. If you're the sort of person who would like to solve the money problem once and for all instead of working for a salary for 40 years, then a startup makes sense.

For a lot of people the conflict is between startups and graduate school. Grad students are just the age, and just the sort of people, to start software startups. You may worry that if you do you'll blow your chances of an academic career. But it's possible to be part of a startup and stay in grad school, especially at first. Two of our three original hackers were in grad school the whole time, and both got their degrees. There are few sources of energy so powerful as a procrastinating grad student.

If you do have to leave grad school, in the worst case it won't be for too long. If a startup fails, it will probably fail quickly enough that you can return to academic life. And if it succeeds, you may find you no longer have such a burning desire to be an assistant professor.

If you want to do it, do it. Starting a startup is not the great mystery it seems from outside. It's not something you have to know about "business" to do. Build something users love, and spend less than you make. How hard is that?

Notes

[1] Google's revenues are about two billion a year, but half comes from ads on other sites.

[2] One advantage startups have over established companies is that there are no discrimination laws about starting businesses. For example, I would be reluctant to start a startup with a woman who had small children, or was likely to have them soon. But you're not allowed to ask prospective employees if they plan to have kids soon. Believe it or not, under current US law, you're not even allowed to discriminate on the basis of intelligence. Whereas when you're starting a company, you can discriminate on any basis you want about who you start it with.

[3] Learning to hack is a lot cheaper than business school, because you can do it mostly on your own. For the price of a Linux box, a copy of K&R, and a few hours of advice from your neighbor's fifteen year old son, you'll be well on your way.

[4] Corollary: Avoid starting a startup to sell things to the biggest company of all, the government. Yes, there are lots of opportunities to sell them technology. But let someone else start those startups.

[5] A friend who started a company in Germany told me they do care about the paperwork there, and that there's more of it. Which helps explain why there are not more startups in Germany.

[6] At the seed stage our valuation was in principle $100,000, because Julian got 10% of the company. But this is a very misleading number, because the money was the least important of the things Julian gave us.

[7] The same goes for companies that seem to want to acquire you. There will be a few that are only pretending to in order to pick your brains. But you can never tell for sure which these are, so the best approach is to seem entirely open, but to fail to mention a few critical technical secrets.

[8] I was as bad an employee as this place was a company. I apologize to anyone who had to work with me there.

[9] You could probably write a book about how to succeed in business by doing everything in exactly the opposite way from the DMV.

Startups in 13 Sentences

Want to start a startup? Get funded by Y Combinator.

Watch how this essay was written.

February 2009

One of the things I always tell startups is a principle I learned from Paul Buchheit: it's better to make a few people really happy than to make a lot of people semi-happy. I was saying recently to a reporter that if I could only tell startups 10 things, this would be one of them. Then I thought: what would the other 9 be?

When I made the list there turned out to be 13:

1. Pick good cofounders.

Cofounders are for a startup what location is for real estate. You can change anything about a house except where it is. In a startup you can change your idea easily, but changing your cofounders is hard. [1] And the success of a startup is almost always a function of its founders.

2. Launch fast.

The reason to launch fast is not so much that it's critical to get your product to market early, but that you haven't really started working on it till you've launched. Launching teaches you what you should have been building. Till you know that you're wasting your time. So the main value of whatever you launch with is as a pretext for engaging users.

3. Let your idea evolve.

This is the second half of launching fast. Launch fast and iterate. It's a big mistake to treat a startup as if it were merely a matter of implementing some brilliant initial idea. As in an essay, most of the ideas appear in the implementing.

4. Understand your users.

You can envision the wealth created by a startup as a rectangle, where one side is the number of users and the other is how much you improve their lives. [2] The second dimension is the one you have most control over. And indeed, the growth in the first will be driven by how well you do in the second. As in science, the hard part is not answering questions but asking them: the hard part is seeing something new that users lack. The better you understand them the better the odds of doing that. That's why so many successful startups make something the founders needed.

5. Better to make a few users love you than a lot ambivalent.

Ideally you want to make large numbers of users love you, but you can't expect to hit that right away. Initially you have to choose between satisfying all the needs of a subset of potential users, or satisfying a subset of the needs of all potential users. Take the first. It's easier to expand userwise than satisfactionwise. And perhaps more importantly, it's harder to lie to yourself. If you think you're 85% of the way to a great product, how do you know it's not 70%? Or 10%? Whereas it's easy to know how many users you have.

6. Offer surprisingly good customer service.

Customers are used to being maltreated. Most of the companies they deal with are quasi-monopolies that get away with atrocious customer service. Your own ideas about what's possible have been unconsciously lowered by such experiences. Try making your customer service not merely good, but surprisingly good. Go out of your way to make people happy. They'll be overwhelmed; you'll see. In the earliest stages of a startup, it pays to offer customer service on a level that wouldn't scale, because it's a way of learning about your users.

7. You make what you measure.

I learned this one from Joe Kraus. [3] Merely measuring something has an uncanny tendency to improve it. If you want to make your user numbers go up, put a big piece of paper on your wall and every day plot the number of users. You'll be delighted when it goes up and disappointed when it goes down. Pretty soon you'll start noticing what makes the number go up, and you'll start to do more of that. Corollary: be careful what you measure.

8. Spend little.

I can't emphasize enough how important it is for a startup to be cheap. Most startups fail before they make something people want, and the most common form of failure is running out of money. So being cheap is (almost) interchangeable with iterating rapidly. [4] But it's more than that. A culture of cheapness keeps companies young in something like the way exercise keeps people young.

9. Get ramen profitable.

"Ramen profitable" means a startup makes just enough to pay the founders' living expenses. It's not rapid prototyping for business models (though it can be), but more a way of hacking the investment process. Once you cross over into ramen profitable, it completely changes your relationship with investors. It's also great for morale.

10. Avoid distractions.

Nothing kills startups like distractions. The worst type are those that pay money: day jobs, consulting, profitable side-projects. The startup may have more long-term potential, but you'll always interrupt working on it to answer calls from people paying you now. Paradoxically, fundraising is this type of distraction, so try to minimize that too.

11. Don't get demoralized.

Though the immediate cause of death in a startup tends to be running out of money, the underlying cause is usually lack of focus. Either the company is run by stupid people (which can't be fixed with advice) or the people are smart but got demoralized. Starting a startup is a huge moral weight. Understand this and make a conscious effort not to be ground down by it, just as you'd be careful to bend at the knees when picking up a heavy box.

12. Don't give up.

Even if you get demoralized, don't give up. You can get surprisingly far by just not giving up. This isn't true in all fields. There are a lot of people who couldn't become good mathematicians no matter how long they persisted. But startups aren't like that. Sheer effort is usually enough, so long as you keep morphing your idea.

13. Deals fall through.

One of the most useful skills we learned from Viaweb was not getting our hopes up. We probably had 20 deals of various types fall through. After the first 10 or so we learned to treat deals as background processes that we should ignore till they terminated. It's very dangerous to morale to start to depend on deals closing, not just because they so often don't, but because it makes them less likely to.

Having gotten it down to 13 sentences, I asked myself which I'd choose if I could only keep one.

Understand your users. That's the key. The essential task in a startup is to create wealth; the dimension of wealth you have most control over is how much you improve users' lives; and the hardest part of that is knowing what to make for them. Once you know what to make, it's mere effort to make it, and most decent hackers are capable of that.

Understanding your users is part of half the principles in this list. That's the reason to launch early, to understand your users. Evolving your idea is the embodiment of understanding your users. Understanding your users well will tend to push you toward making something that makes a few people deeply happy. The most important reason for having surprisingly good customer service is that it helps you understand your users. And understanding your users will even ensure your morale, because when everything else is collapsing around you, having just ten users who love you will keep you going.

Notes

[1] Strictly speaking it's impossible without a time machine.

[2] In practice it's more like a ragged comb.

[3] Joe thinks one of the founders of Hewlett Packard said it first, but he doesn't remember which.

[4] They'd be interchangeable if markets stood still. Since they don't, working twice as fast is better than having twice as much time.

Hiring is Obsolete

Want to start a startup? Get funded by Y Combinator.

May 2005

(This essay is derived from a talk at the Berkeley CSUA.)

The three big powers on the Internet now are Yahoo, Google, and Microsoft. Average age of their founders: 24. So it is pretty well established now that grad students can start successful companies. And if grad students can do it, why not undergrads?

Like everything else in technology, the cost of starting a startup has decreased dramatically. Now it's so low that it has disappeared into the noise. The main cost of starting a Web-based startup is food and rent. Which means it doesn't cost much more to start a company than to be a total slacker. You can probably start a startup on ten thousand dollars of seed funding, if you're prepared to live on ramen.

The less it costs to start a company, the less you need the permission of investors to do it. So a lot of people will be able to start companies now who never could have before.

The most interesting subset may be those in their early twenties. I'm not so excited about founders who have everything investors want except intelligence, or everything except energy. The most promising group to be liberated by the new, lower threshold are those who have everything investors want except experience.

Market Rate

I once claimed that nerds were unpopular in secondary school mainly because they had better things to do than work full-time at being popular. Some said I was just telling people what they wanted to hear. Well, I'm now about to do that in a spectacular way: I think undergraduates are undervalued.

Or more precisely, I think few realize the huge spread in the value of 20 year olds. Some, it's true, are not very capable. But others are more capable than all but a handful of 30 year olds. [1]

Till now the problem has always been that it's difficult to pick them out. Every VC in the world, if they could go back in time, would try to invest in Microsoft. But which would have then? How many would have understood that this particular 19 year old was Bill Gates?

It's hard to judge the young because (a) they change rapidly, (b) there is great variation between them, and (c) they're individually inconsistent. That last one is a big problem. When you're young, you occasionally say and do stupid things even when you're smart. So if the algorithm is to filter out people who say stupid things, as many investors and employers unconsciously do, you're going to get a lot of false positives.

Most organizations who hire people right out of college are only aware of the average value of 22 year olds, which is not that high. And so the idea for most of the twentieth century was that everyone had to begin as a trainee in some entry-level job. Organizations realized there was a lot of variation in the incoming stream, but instead of pursuing this thought they tended to suppress it, in the belief that it was good for even the most promising kids to start at the bottom, so they didn't get swelled heads.

The most productive young people will always be undervalued by large organizations, because the young have no performance to measure yet, and any error in guessing their ability will tend toward the mean.

What's an especially productive 22 year old to do? One thing you can do is go over the heads of organizations, directly to the users. Any company that hires you is, economically, acting as a proxy for the customer. The rate at which they value you (though they may not consciously realize it) is an attempt to guess your value to the user. But there's a way to appeal their judgement. If you want, you can opt to be valued directly by users, by starting your own company.

The market is a lot more discerning than any employer. And it is completely non-discriminatory. On the Internet, nobody knows you're a dog. And more to the point, nobody knows you're 22. All users care about is whether your site or software gives them what they want. They don't care if the person behind it is a high school kid.

If you're really productive, why not make employers pay market rate for you? Why go work as an ordinary employee for a big company, when you could start a startup and make them buy it to get you?

When most people hear the word "startup," they think of the famous ones that have gone public. But most startups that succeed do it by getting bought. And usually the acquirer doesn't just want the technology, but the people who created it as well.

Often big companies buy startups before they're profitable. Obviously in such cases they're not after revenues. What they want is the development team and the software they've built so far. When a startup gets bought for 2 or 3 million six months in, it's really more of a hiring bonus than an acquisition.

I think this sort of thing will happen more and more, and that it will be better for everyone. It's obviously better for the people who start the startup, because they get a big chunk of money up front. But I think it will be better for the acquirers too. The central problem in big companies, and the main reason they're so much less productive than small companies, is the difficulty of valuing each person's work. Buying larval startups solves that problem for them: the acquirer doesn't pay till the developers have proven themselves. Acquirers are protected on the downside, but still get most of the upside.

Product Development

Buying startups also solves another problem afflicting big companies: they can't do product development. Big companies are good at extracting the value from existing products, but bad at creating new ones.

Why? It's worth studying this phenomenon in detail, because this is the raison d'etre of startups.

To start with, most big companies have some kind of turf to protect, and this tends to warp their development decisions. For example, Web-based applications are hot now, but within Microsoft there must be a lot of ambivalence about them, because the very idea of Web-based software threatens the desktop. So any Web-based application that Microsoft ends up with, will probably, like Hotmail, be something developed outside the company.

Another reason big companies are bad at developing new products is that the kind of people who do that tend not to have much power in big companies (unless they happen to be the CEO). Disruptive technologies are developed by disruptive people. And they either don't work for the big company, or have been outmaneuvered by yes-men and have comparatively little influence.

Big companies also lose because they usually only build one of each thing. When you only have one Web browser, you can't do anything really risky with it. If ten different startups design ten different Web browsers and you take the best, you'll probably get something better.

The more general version of this problem is that there are too many new ideas for companies to explore them all. There might be 500 startups right now who think they're making something Microsoft might buy. Even Microsoft probably couldn't manage 500 development projects in-house.

Big companies also don't pay people the right way. People developing a new product at a big company get paid roughly the same whether it succeeds or fails. People at a startup expect to get rich if the product succeeds, and get nothing if it fails. [2] So naturally the people at the startup work a lot harder.

The mere bigness of big companies is an obstacle. In startups, developers are often forced to talk directly to users, whether they want to or not, because there is no one else to do sales and support. It's painful doing sales, but you learn much more from trying to sell people something than reading what they said in focus groups.

And then of course, big companies are bad at product development because they're bad at everything. Everything happens slower in big companies than small ones, and product development is something that has to happen fast, because you have to go through a lot of iterations to get something good.

Trend

I think the trend of big companies buying startups will only accelerate. One of the biggest remaining obstacles is pride. Most companies, at least unconsciously, feel they ought to be able to develop stuff in house, and that buying startups is to some degree an admission of failure. And so, as people generally do with admissions of failure, they put it off for as long as possible. That makes the acquisition very expensive when it finally happens.

What companies should do is go out and discover startups when they're young, before VCs have puffed them up into something that costs hundreds of millions to acquire. Much of what VCs add, the acquirer doesn't need anyway.

Why don't acquirers try to predict the companies they're going to have to buy for hundreds of millions, and grab them early for a tenth or a twentieth of that? Because they can't predict the winners in advance? If they're only paying a twentieth as much, they only have to predict a twentieth as well. Surely they can manage that.

I think companies that acquire technology will gradually learn to go after earlier stage startups. They won't necessarily buy them outright. The solution may be some hybrid of investment and acquisition: for example, to buy a chunk of the company and get an option to buy the rest later.

When companies buy startups, they're effectively fusing recruiting and product development. And I think that's more efficient than doing the two separately, because you always get people who are really committed to what they're working on.

Plus this method yields teams of developers who already work well together. Any conflicts between them have been ironed out under the very hot iron of running a startup. By the time the acquirer gets them, they're finishing one another's sentences. That's valuable in software, because so many bugs occur at the boundaries between different people's code.

Investors

The increasing cheapness of starting a company doesn't just give hackers more power relative to employers. It also gives them more power relative to investors.

The conventional wisdom among VCs is that hackers shouldn't be allowed to run their own companies. The founders are supposed to accept MBAs as their bosses, and themselves take on some title like Chief Technical Officer. There may be cases where this is a good idea. But I think founders will increasingly be able to push back in the matter of control, because they just don't need the investors' money as much as they used to.

Startups are a comparatively new phenomenon. Fairchild Semiconductor is considered the first VC-backed startup, and they were founded in 1959, less than fifty years ago. Measured on the time scale of social change, what we have now is pre-beta. So we shouldn't assume the way startups work now is the way they have to work.

Fairchild needed a lot of money to get started. They had to build actual factories. What does the first round of venture funding for a Web-based startup get spent on today? More money can't get software written faster; it isn't needed for facilities, because those can now be quite cheap; all money can really buy you is sales and marketing. A sales force is worth something, I'll admit. But marketing is increasingly irrelevant. On the Internet, anything genuinely good will spread by word of mouth.

Investors' power comes from money. When startups need less money, investors have less power over them. So future founders may not have to accept new CEOs if they don't want them. The VCs will have to be dragged kicking and screaming down this road, but like many things people have to be dragged kicking and screaming toward, it may actually be good for them.

Google is a sign of the way things are going. As a condition of funding, their investors insisted they hire someone old and experienced as CEO. But from what I've heard the founders didn't just give in and take whoever the VCs wanted. They delayed for an entire year, and when they did finally take a CEO, they chose a guy with a PhD in computer science.

It sounds to me as if the founders are still the most powerful people in the company, and judging by Google's performance, their youth and inexperience doesn't seem to have hurt them. Indeed, I suspect Google has done better than they would have if the founders had given the VCs what they wanted, when they wanted it, and let some MBA take over as soon as they got their first round of funding.

I'm not claiming the business guys installed by VCs have no value. Certainly they have. But they don't need to become the founders' bosses, which is what that title CEO means. I predict that in the future the executives installed by VCs will increasingly be COOs rather than CEOs. The founders will run engineering directly, and the rest of the company through the COO.

The Open Cage

With both employers and investors, the balance of power is slowly shifting towards the young. And yet they seem the last to realize it. Only the most ambitious undergrads even consider starting their own company when they graduate. Most just want to get a job.

Maybe this is as it should be. Maybe if the idea of starting a startup is intimidating, you filter out the uncommitted. But I suspect the filter is set a little too high. I think there are people who could, if they tried, start successful startups, and who instead let themselves be swept into the intake ducts of big companies.

Have you ever noticed that when animals are let out of cages, they don't always realize at first that the door's open? Often they have to be poked with a stick to get them out. Something similar happened with blogs. People could have been publishing online in 1995, and yet blogging has only really taken off in the last couple years. In 1995 we thought only professional writers were entitled to publish their ideas, and that anyone else who did was a crank. Now publishing online is becoming so popular that everyone wants to do it, even print journalists. But blogging has not taken off recently because of any technical innovation; it just took eight years for everyone to realize the cage was open.

I think most undergrads don't realize yet that the economic cage is open. A lot have been told by their parents that the route to success is to get a good job. This was true when their parents were in college, but it's less true now. The route to success is to build something valuable, and you don't have to be working for an existing company to do that. Indeed, you can often do it better if you're not.

When I talk to undergrads, what surprises me most about them is how conservative they are. Not politically, of course. I mean they don't seem to want to take risks. This is a mistake, because the younger you are, the more risk you can take.

Risk

Risk and reward are always proportionate. For example, stocks are riskier than bonds, and over time always have greater returns. So why does anyone invest in bonds? The catch is that phrase "over time." Stocks will generate greater returns over thirty years, but they might lose value from year to year. So what you should invest in depends on how soon you need the money. If you're young, you should take the riskiest investments you can find.

All this talk about investing may seem very theoretical. Most undergrads probably have more debts than assets. They may feel they have nothing to invest. But that's not true: they have their time to invest, and the same rule about risk applies there. Your early twenties are exactly the time to take insane career risks.

The reason risk is always proportionate to reward is that market forces make it so. People will pay extra for stability. So if you choose stability-- by buying bonds, or by going to work for a big company-- it's going to cost you.

Riskier career moves pay better on average, because there is less demand for them. Extreme choices like starting a startup are so frightening that most people won't even try. So you don't end up having as much competition as you might expect, considering the prizes at stake.

The math is brutal. While perhaps 9 out of 10 startups fail, the one that succeeds will pay the founders more than 10 times what they would have made in an ordinary job. [3] That's the sense in which startups pay better "on average."

Remember that. If you start a startup, you'll probably fail. Most startups fail. It's the nature of the business. But it's not necessarily a mistake to try something that has a 90% chance of failing, if you can afford the risk. Failing at 40, when you have a family to support, could be serious. But if you fail at 22, so what? If you try to start a startup right out of college and it tanks, you'll end up at 23 broke and a lot smarter. Which, if you think about it, is roughly what you hope to get from a graduate program.

Even if your startup does tank, you won't harm your prospects with employers. To make sure I asked some friends who work for big companies. I asked managers at Yahoo, Google, Amazon, Cisco and Microsoft how they'd feel about two candidates, both 24, with equal ability, one who'd tried to start a startup that tanked, and another who'd spent the two years since college working as a developer at a big company. Every one responded that they'd prefer the guy who'd tried to start his own company. Zod Nazem, who's in charge of engineering at Yahoo, said:

I actually put more value on the guy with the failed startup. And you can quote me!

So there you have it. Want to get hired by Yahoo? Start your own company.

The Man is the Customer

If even big employers think highly of young hackers who start companies, why don't more do it? Why are undergrads so conservative? I think it's because they've spent so much time in institutions.

The first twenty years of everyone's life consists of being piped from one institution to another. You probably didn't have much choice about the secondary schools you went to. And after high school it was probably understood that you were supposed to go to college. You may have had a few different colleges to choose between, but they were probably pretty similar. So by this point you've been riding on a subway line for twenty years, and the next stop seems to be a job.

Actually college is where the line ends. Superficially, going to work for a company may feel like just the next in a series of institutions, but underneath, everything is different. The end of school is the fulcrum of your life, the point where you go from net consumer to net producer.

The other big change is that now, you're steering. You can go anywhere you want. So it may be worth standing back and understanding what's going on, instead of just doing the default thing.

All through college, and probably long before that, most undergrads have been thinking about what employers want. But what really matters is what customers want, because they're the ones who give employers the money to pay you.

So instead of thinking about what employers want, you're probably better off thinking directly about what users want. To the extent there's any difference between the two, you can even use that to your advantage if you start a company of your own. For example, big companies like docile conformists. But this is merely an artifact of their bigness, not something customers need.

Grad School

I didn't consciously realize all this when I was graduating from college-- partly because I went straight to grad school. Grad school can be a pretty good deal, even if you think of one day starting a startup. You can start one when you're done, or even pull the ripcord part way through, like the founders of Yahoo and Google.

Grad school makes a good launch pad for startups, because you're collected together with a lot of smart people, and you have bigger chunks of time to work on your own projects than an undergrad or corporate employee would. As long as you have a fairly tolerant advisor, you can take your time developing an idea before turning it into a company. David Filo and Jerry Yang started the Yahoo directory in February 1994 and were getting a million hits a day by the fall, but they didn't actually drop out of grad school and start a company till March 1995.

You could also try the startup first, and if it doesn't work, then go to grad school. When startups tank they usually do it fairly quickly. Within a year you'll know if you're wasting your time.

If it fails, that is. If it succeeds, you may have to delay grad school a little longer. But you'll have a much more enjoyable life once there than you would on a regular grad student stipend.

Experience

Another reason people in their early twenties don't start startups is that they feel they don't have enough experience. Most investors feel the same.

I remember hearing a lot of that word "experience" when I was in college. What do people really mean by it? Obviously it's not the experience itself that's valuable, but something it changes in your brain. What's different about your brain after you have "experience," and can you make that change happen faster?

I now have some data on this, and I can tell you what tends to be missing when people lack experience. I've said that every startup needs three things: to start with good people, to make something users want, and not to spend too much money. It's the middle one you get wrong when you're inexperienced. There are plenty of undergrads with enough technical skill to write good software, and undergrads are not especially prone to waste money. If they get something wrong, it's usually not realizing they have to make something people want.

This is not exclusively a failing of the young. It's common for startup founders of all ages to build things no one wants.

Fortunately, this flaw should be easy to fix. If undergrads were all bad programmers, the problem would be a lot harder. It can take years to learn how to program. But I don't think it takes years to learn how to make things people want. My hypothesis is that all you have to do is smack hackers on the side of the head and tell them: Wake up. Don't sit here making up a priori theories about what users need. Go find some users and see what they need.

Most successful startups not only do something very specific, but solve a problem people already know they have.

The big change that "experience" causes in your brain is learning that you need to solve people's problems. Once you grasp that, you advance quickly to the next step, which is figuring out what those problems are. And that takes some effort, because the way software actually gets used, especially by the people who pay the most for it, is not at all what you might expect. For example, the stated purpose of Powerpoint is to present ideas. Its real role is to overcome people's fear of public speaking. It allows you to give an impressive-looking talk about nothing, and it causes the audience to sit in a dark room looking at slides, instead of a bright one looking at you.

This kind of thing is out there for anyone to see. The key is to know to look for it-- to realize that having an idea for a startup is not like having an idea for a class project. The goal in a startup is not to write a cool piece of software. It's to make something people want. And to do that you have to look at users-- forget about hacking, and just look at users. This can be quite a mental adjustment, because little if any of the software you write in school even has users.

A few steps before a Rubik's Cube is solved, it still looks like a mess. I think there are a lot of undergrads whose brains are in a similar position: they're only a few steps away from being able to start successful startups, if they wanted to, but they don't realize it. They have more than enough technical skill. They just haven't realized yet that the way to create wealth is to make what users want, and that employers are just proxies for users in which risk is pooled.

If you're young and smart, you don't need either of those. You don't need someone else to tell you what users want, because you can figure it out yourself. And you don't want to pool risk, because the younger you are, the more risk you should take.

A Public Service Message

I'd like to conclude with a joint message from me and your parents. Don't drop out of college to start a startup. There's no rush. There will be plenty of time to start companies after you graduate. In fact, it may be just as well to go work for an existing company for a couple years after you graduate, to learn how companies work.

And yet, when I think about it, I can't imagine telling Bill Gates at 19 that he should wait till he graduated to start a company. He'd have told me to get lost. And could I have honestly claimed that he was harming his future-- that he

was learning less by working at ground zero of the microcomputer revolution than he would have if he'd been taking classes back at Harvard? No, probably not.

And yes, while it is probably true that you'll learn some valuable things by going to work for an existing company for a couple years before starting your own, you'd learn a thing or two running your own company during that time too.

The advice about going to work for someone else would get an even colder reception from the 19 year old Bill Gates. So I'm supposed to finish college, then go work for another company for two years, and then I can start my own? I have to wait till I'm 23? That's four years. That's more than twenty percent of my life so far. Plus in four years it will be way too late to make money writing a Basic interpreter for the Altair.

And he'd be right. The Apple II was launched just two years later. In fact, if Bill had finished college and gone to work for another company as we're suggesting, he might well have gone to work for Apple. And while that would probably have been better for all of us, it wouldn't have been better for him.

So while I stand by our responsible advice to finish college and then go work for a while before starting a startup, I have to admit it's one of those things the old tell the young, but don't expect them to listen to. We say this sort of thing mainly so we can claim we warned you. So don't say I didn't warn you.

Notes

[1] The average B-17 pilot in World War II was in his early twenties. (Thanks to Tad Marko for pointing this out.)

[2] If a company tried to pay employees this way, they'd be called unfair. And yet when they buy some startups and not others, no one thinks of calling that unfair.

[3] The 1/10 success rate for startups is a bit of an urban legend. It's suspiciously neat. My guess is the odds are slightly worse.

How to Make Wealth

Want to start a startup? Get funded by Y Combinator.

May 2004

(This essay was originally published in Hackers & Painters.)

If you wanted to get rich, how would you do it? I think your best bet would be to start or join a startup. That's been a reliable way to get rich for hundreds of years. The word "startup" dates from the 1960s, but what happens in one is very similar to the venture-backed trading voyages of the Middle Ages.

Startups usually involve technology, so much so that the phrase "high-tech startup" is almost redundant. A startup is a small company that takes on a hard technical problem.

Lots of people get rich knowing nothing more than that. You don't have to know physics to be a good pitcher. But I think it could give you an edge to understand the underlying principles. Why do startups have to be small? Will a startup inevitably stop being a startup as it grows larger? And why do they so often work on developing new technology? Why are there so many startups selling new drugs or computer software, and none selling corn oil or laundry detergent?

The Proposition

Economically, you can think of a startup as a way to compress your whole working life into a few years. Instead of working at a low intensity for forty years, you work as hard as you possibly can for four. This pays especially well in technology, where you earn a premium for working fast.

Here is a brief sketch of the economic proposition. If you're a good hacker in your mid twenties, you can get a job paying about $80,000 per year. So on average such a hacker must be able to do at least $80,000 worth of work per year for the company just to break even. You could probably work twice as many hours as a corporate employee, and if you focus you can probably get three times as much done in an hour. [1] You should get another multiple of two, at least, by eliminating the drag of the pointy-haired middle manager who would be your boss in a big company. Then there is one more multiple: how much smarter are you than your job description expects you to be? Suppose another multiple of three. Combine all these multipliers, and I'm claiming you could be 36 times more productive than you're expected to be in a random corporate job. [2] If a fairly good hacker is worth $80,000 a year at a big company, then a smart hacker working very hard without any corporate bullshit to slow him down should be able to do work worth about $3 million a year.

Like all back-of-the-envelope calculations, this one has a lot of wiggle room. I wouldn't try to defend the actual numbers. But I stand by the structure of the calculation. I'm not claiming the multiplier is precisely 36, but it is certainly more than 10, and probably rarely as high as 100.

If $3 million a year seems high, remember that we're talking about the limit case: the case where you not only have zero leisure time but indeed work so hard that you endanger your health.

Startups are not magic. They don't change the laws of wealth creation. They just represent a point at the far end of the curve. There is a conservation law at work here: if you want to make a million dollars, you have to endure a million dollars' worth of pain. For example, one way to make a million dollars would be to work for the Post Office your whole life, and save every penny of your salary. Imagine the stress of working for the Post Office for fifty years. In a startup you compress all this stress into three or four years. You do tend to get a certain bulk discount if you buy the economy-size pain, but you can't evade the fundamental conservation law. If starting a startup were easy, everyone would do it.

Millions, not Billions

If $3 million a year seems high to some people, it will seem low to others. Three million? How do I get to be a billionaire, like Bill Gates?

So let's get Bill Gates out of the way right now. It's not a good idea to use famous rich people as examples, because the press only write about the very richest, and these tend to be outliers. Bill Gates is a smart, determined, and hardworking man, but you need more than that to make as much money as he has. You also need to be very lucky.

There is a large random factor in the success of any company. So the guys you end up reading about in the papers are the ones who are very smart, totally dedicated, and win the lottery. Certainly Bill is smart and dedicated, but Microsoft also happens to have been the beneficiary of one of the most spectacular blunders in the history of business: the licensing deal for DOS. No doubt Bill did everything he could to steer IBM into making that blunder, and he has done an excellent job of exploiting it, but if there had been one person with a brain on IBM's side, Microsoft's future would have been very different. Microsoft at that stage had little leverage over IBM. They were effectively a component supplier. If IBM had required an exclusive license, as they should have, Microsoft would still have signed the deal. It would still have meant a lot of money for them, and IBM could easily have gotten an operating system elsewhere.

Instead IBM ended up using all its power in the market to give Microsoft control of the PC standard. From that point, all Microsoft had to do was execute. They never had to bet the company on a bold decision. All they had to do was play hardball with licensees and copy more innovative products reasonably promptly.

If IBM hadn't made this mistake, Microsoft would still have been a successful company, but it could not have grown so big so fast. Bill Gates would be rich, but he'd be somewhere near the bottom of the Forbes 400 with the other guys his age.

There are a lot of ways to get rich, and this essay is about only one of them. This essay is about how to make money by creating wealth and getting paid for it. There are plenty of other ways to get money, including chance, speculation, marriage, inheritance, theft, extortion, fraud, monopoly, graft, lobbying, counterfeiting, and prospecting. Most of the greatest fortunes have probably involved several of these.

The advantage of creating wealth, as a way to get rich, is not just that it's more legitimate (many of the other methods are now illegal) but that it's more straightforward. You just have to do something people want.

Money Is Not Wealth

If you want to create wealth, it will help to understand what it is. Wealth is not the same thing as money. [3] Wealth is as old as human history. Far older, in fact; ants have wealth. Money is a comparatively recent invention.

Wealth is the fundamental thing. Wealth is stuff we want: food, clothes, houses, cars, gadgets, travel to interesting places, and so on. You can have wealth without having money. If you had a magic machine that could on command make you a car or cook you dinner or do your laundry, or do anything else you wanted, you wouldn't need money. Whereas if you were in the middle of Antarctica, where there is nothing to buy, it wouldn't matter how much money you had.

Wealth is what you want, not money. But if wealth is the important thing, why does everyone talk about making money? It is a kind of shorthand: money is a way of moving wealth, and in practice they are usually interchangeable. But they are not the same thing, and unless you plan to get rich by counterfeiting, talking about making money can make it harder to understand how to make money.

Money is a side effect of specialization. In a specialized society, most of the things you need, you can't make for yourself. If you want a potato or a pencil or a place to live, you have to get it from someone else.

How do you get the person who grows the potatoes to give you some? By giving him something he wants in return. But you can't get very far by trading things directly with the people who need them. If you make violins, and none of the local farmers wants one, how will you eat?

The solution societies find, as they get more specialized, is to make the trade into a two-step process. Instead of trading violins directly for potatoes, you trade violins for, say, silver, which you can then trade again for anything else you need. The intermediate stuff-- the medium of exchange-- can be anything that's rare and portable. Historically metals have been the most common, but recently we've been using a medium of exchange, called the dollar, that

doesn't physically exist. It works as a medium of exchange, however, because its rarity is guaranteed by the U.S. Government.

The advantage of a medium of exchange is that it makes trade work. The disadvantage is that it tends to obscure what trade really means. People think that what a business does is make money. But money is just the intermediate stage-- just a shorthand-- for whatever people want. What most businesses really do is make wealth. They do something people want. [4]

The Pie Fallacy

A surprising number of people retain from childhood the idea that there is a fixed amount of wealth in the world. There is, in any normal family, a fixed amount of money at any moment. But that's not the same thing.

When wealth is talked about in this context, it is often described as a pie. "You can't make the pie larger," say politicians. When you're talking about the amount of money in one family's bank account, or the amount available to a government from one year's tax revenue, this is true. If one person gets more, someone else has to get less.

I can remember believing, as a child, that if a few rich people had all the money, it left less for everyone else. Many people seem to continue to believe something like this well into adulthood. This fallacy is usually there in the background when you hear someone talking about how x percent of the population have y percent of the wealth. If you plan to start a startup, then whether you realize it or not, you're planning to disprove the Pie Fallacy.

What leads people astray here is the abstraction of money. Money is not wealth. It's just something we use to move wealth around. So although there may be, in certain specific moments (like your family, this month) a fixed amount of money available to trade with other people for things you want, there is not a fixed amount of wealth in the world. You can make more wealth. Wealth has been getting created and destroyed (but on balance, created) for all of human history.

Suppose you own a beat-up old car. Instead of sitting on your butt next summer, you could spend the time restoring your car to pristine condition. In doing so you create wealth. The world is-- and you specifically are-- one pristine old car the richer. And not just in some metaphorical way. If you sell your car, you'll get more for it.

In restoring your old car you have made yourself richer. You haven't made anyone else poorer. So there is obviously not a fixed pie. And in fact, when you look at it this way, you wonder why anyone would think there was. [5]

Kids know, without knowing they know, that they can create wealth. If you need to give someone a present and don't have any money, you make one. But kids are so bad at making things that they consider home-made presents to be a distinct, inferior, sort of thing to store-bought ones-- a mere expression of the proverbial thought that counts. And indeed, the lumpy ashtrays we made for our parents did not have much of a resale market.

Craftsmen

The people most likely to grasp that wealth can be created are the ones who are good at making things, the craftsmen. Their hand-made objects become store-bought ones. But with the rise of industrialization there are fewer and fewer craftsmen. One of the biggest remaining groups is computer programmers.

A programmer can sit down in front of a computer and create wealth. A good piece of software is, in itself, a valuable thing. There is no manufacturing to confuse the issue. Those characters you type are a complete, finished product. If someone sat down and wrote a web browser that didn't suck (a fine idea, by the way), the world would be that much richer. [5b]

Everyone in a company works together to create wealth, in the sense of making more things people want. Many of the employees (e.g. the people in the mailroom or the personnel department) work at one remove from the actual making of stuff. Not the programmers. They literally think the product, one line at a time. And so it's clearer to programmers that wealth is something that's made, rather than being distributed, like slices of a pie, by some imaginary Daddy.

It's also obvious to programmers that there are huge variations in the rate at which wealth is created. At Viaweb we had one programmer who was a sort of monster of productivity. I remember watching what he did one long day and estimating that he had added several hundred thousand dollars to the market value of the company. A great programmer, on a roll, could create a million dollars worth of wealth in a couple weeks. A mediocre programmer over the same period will generate zero or even negative wealth (e.g. by introducing bugs).

This is why so many of the best programmers are libertarians. In our world, you sink or swim, and there are no excuses. When those far removed from the creation of wealth-- undergraduates, reporters, politicians-- hear that the richest 5% of the people have half the total wealth, they tend to think injustice! An experienced programmer would be more likely to think is that all? The top 5% of programmers probably write 99% of the good software.

Wealth can be created without being sold. Scientists, till recently at least, effectively donated the wealth they created. We are all richer for knowing about penicillin, because we're less likely to die from infections. Wealth is whatever people want, and not dying is certainly something we want. Hackers often donate their work by writing open source software that anyone can use for free. I am much the richer for the operating system FreeBSD, which I'm running on the computer I'm using now, and so is Yahoo, which runs it on all their servers.

What a Job Is

In industrialized countries, people belong to one institution or another at least until their twenties. After all those years you get used to the idea of belonging to a group of people who all get up in the morning, go to some set of buildings, and do things that they do not, ordinarily, enjoy doing. Belonging to such a group becomes part of your

identity: name, age, role, institution. If you have to introduce yourself, or someone else describes you, it will be as something like, John Smith, age 10, a student at such and such elementary school, or John Smith, age 20, a student at such and such college.

When John Smith finishes school he is expected to get a job. And what getting a job seems to mean is joining another institution. Superficially it's a lot like college. You pick the companies you want to work for and apply to join them. If one likes you, you become a member of this new group. You get up in the morning and go to a new set of buildings, and do things that you do not, ordinarily, enjoy doing. There are a few differences: life is not as much fun, and you get paid, instead of paying, as you did in college. But the similarities feel greater than the differences. John Smith is now John Smith, 22, a software developer at such and such corporation.

In fact John Smith's life has changed more than he realizes. Socially, a company looks much like college, but the deeper you go into the underlying reality, the more different it gets.

What a company does, and has to do if it wants to continue to exist, is earn money. And the way most companies make money is by creating wealth. Companies can be so specialized that this similarity is concealed, but it is not only manufacturing companies that create wealth. A big component of wealth is location. Remember that magic machine that could make you cars and cook you dinner and so on? It would not be so useful if it delivered your dinner to a random location in central Asia. If wealth means what people want, companies that move things also create wealth. Ditto for many other kinds of companies that don't make anything physical. Nearly all companies exist to do something people want.

And that's what you do, as well, when you go to work for a company. But here there is another layer that tends to obscure the underlying reality. In a company, the work you do is averaged together with a lot of other people's. You may not even be aware you're doing something people want. Your contribution may be indirect. But the company as a whole must be giving people something they want, or they won't make any money. And if they are paying you x dollars a year, then on average you must be contributing at least x dollars a year worth of work, or the company will be spending more than it makes, and will go out of business.

Someone graduating from college thinks, and is told, that he needs to get a job, as if the important thing were becoming a member of an institution. A more direct way to put it would be: you need to start doing something people want. You don't need to join a company to do that. All a company is is a group of people working together to do something people want. It's doing something people want that matters, not joining the group. [6]

For most people the best plan probably is to go to work for some existing company. But it is a good idea to understand what's happening when you do this. A job means doing something people want, averaged together with everyone else in that company.

Working Harder

That averaging gets to be a problem. I think the single biggest problem afflicting large companies is the difficulty of assigning a value to each person's work. For the most part they punt. In a big company you get paid a fairly predictable salary for working fairly hard. You're expected not to be obviously incompetent or lazy, but you're not expected to devote your whole life to your work.

It turns out, though, that there are economies of scale in how much of your life you devote to your work. In the right kind of business, someone who really devoted himself to work could generate ten or even a hundred times as much wealth as an average employee. A programmer, for example, instead of chugging along maintaining and updating an existing piece of software, could write a whole new piece of software, and with it create a new source of revenue.

Companies are not set up to reward people who want to do this. You can't go to your boss and say, I'd like to start working ten times as hard, so will you please pay me ten times as much? For one thing, the official fiction is that you are already working as hard as you can. But a more serious problem is that the company has no way of measuring the value of your work.

Salesmen are an exception. It's easy to measure how much revenue they generate, and they're usually paid a percentage of it. If a salesman wants to work harder, he can just start doing it, and he will automatically get paid proportionally more.

There is one other job besides sales where big companies can hire first-rate people: in the top management jobs. And for the same reason: their performance can be measured. The top managers are held responsible for the performance of the entire company. Because an ordinary employee's performance can't usually be measured, he is not expected to do more than put in a solid effort. Whereas top management, like salespeople, have to actually come up with the numbers. The CEO of a company that tanks cannot plead that he put in a solid effort. If the company does badly, he's done badly.

A company that could pay all its employees so straightforwardly would be enormously successful. Many employees would work harder if they could get paid for it. More importantly, such a company would attract people who wanted to work especially hard. It would crush its competitors.

Unfortunately, companies can't pay everyone like salesmen. Salesmen work alone. Most employees' work is tangled together. Suppose a company makes some kind of consumer gadget. The engineers build a reliable gadget with all kinds of new features; the industrial designers design a beautiful case for it; and then the marketing people convince everyone that it's something they've got to have. How do you know how much of the gadget's sales are due to each group's efforts? Or, for that matter, how much is due to the creators of past gadgets that gave the company a reputation for quality? There's no way to untangle all their contributions. Even if you could read the minds of the consumers, you'd find these factors were all blurred together.

If you want to go faster, it's a problem to have your work tangled together with a large number of other people's. In a large group, your performance is not separately measurable-- and the rest of the group slows you down.

Measurement and Leverage

To get rich you need to get yourself in a situation with two things, measurement and leverage. You need to be in a position where your performance can be measured, or there is no way to get paid more by doing more. And you have to have leverage, in the sense that the decisions you make have a big effect.

Measurement alone is not enough. An example of a job with measurement but not leverage is doing piecework in a sweatshop. Your performance is measured and you get paid accordingly, but you have no scope for decisions. The only decision you get to make is how fast you work, and that can probably only increase your earnings by a factor of two or three.

An example of a job with both measurement and leverage would be lead actor in a movie. Your performance can be measured in the gross of the movie. And you have leverage in the sense that your performance can make or break it.

CEOs also have both measurement and leverage. They're measured, in that the performance of the company is their performance. And they have leverage in that their decisions set the whole company moving in one direction or another.

I think everyone who gets rich by their own efforts will be found to be in a situation with measurement and leverage. Everyone I can think of does: CEOs, movie stars, hedge fund managers, professional athletes. A good hint to the presence of leverage is the possibility of failure. Upside must be balanced by downside, so if there is big potential for gain there must also be a terrifying possibility of loss. CEOs, stars, fund managers, and athletes all live with the sword hanging over their heads; the moment they start to suck, they're out. If you're in a job that feels safe, you are not going to get rich, because if there is no danger there is almost certainly no leverage.

But you don't have to become a CEO or a movie star to be in a situation with measurement and leverage. All you need to do is be part of a small group working on a hard problem.

Smallness = Measurement

If you can't measure the value of the work done by individual employees, you can get close. You can measure the value of the work done by small groups.

One level at which you can accurately measure the revenue generated by employees is at the level of the whole company. When the company is small, you are thereby fairly close to measuring the contributions of individual employees. A viable startup might only have ten employees, which puts you within a factor of ten of measuring individual effort.

Starting or joining a startup is thus as close as most people can get to saying to one's boss, I want to work ten times as hard, so please pay me ten times as much. There are two differences: you're not saying it to your boss, but directly to the customers (for whom your boss is only a proxy after all), and you're not doing it individually, but along with a small group of other ambitious people.

It will, ordinarily, be a group. Except in a few unusual kinds of work, like acting or writing books, you can't be a company of one person. And the people you work with had better be good, because it's their work that yours is going to be averaged with.

A big company is like a giant galley driven by a thousand rowers. Two things keep the speed of the galley down. One is that individual rowers don't see any result from working harder. The other is that, in a group of a thousand people, the average rower is likely to be pretty average.

If you took ten people at random out of the big galley and put them in a boat by themselves, they could probably go faster. They would have both carrot and stick to motivate them. An energetic rower would be encouraged by the thought that he could have a visible effect on the speed of the boat. And if someone was lazy, the others would be more likely to notice and complain.

But the real advantage of the ten-man boat shows when you take the ten best rowers out of the big galley and put them in a boat together. They will have all the extra motivation that comes from being in a small group. But more importantly, by selecting that small a group you can get the best rowers. Each one will be in the top 1%. It's a much better deal for them to average their work together with a small group of their peers than to average it with everyone.

That's the real point of startups. Ideally, you are getting together with a group of other people who also want to work a lot harder, and get paid a lot more, than they would in a big company. And because startups tend to get founded by self-selecting groups of ambitious people who already know one another (at least by reputation), the level of measurement is more precise than you get from smallness alone. A startup is not merely ten people, but ten people like you.

Steve Jobs once said that the success or failure of a startup depends on the first ten employees. I agree. If anything, it's more like the first five. Being small is not, in itself, what makes startups kick butt, but rather that small groups can be select. You don't want small in the sense of a village, but small in the sense of an all-star team.

The larger a group, the closer its average member will be to the average for the population as a whole. So all other things being equal, a very able person in a big company is probably getting a bad deal, because his performance is dragged down by the overall lower performance of the others. Of course, all other things often are not equal: the able person may not care about money, or may prefer the stability of a large company. But a very able person who does care about money will ordinarily do better to go off and work with a small group of peers.

Technology = Leverage

Startups offer anyone a way to be in a situation with measurement and leverage. They allow measurement because they're small, and they offer leverage because they make money by inventing new technology.

What is technology? It's technique. It's the way we all do things. And when you discover a new way to do things, its value is multiplied by all the people who use it. It is the proverbial fishing rod, rather than the fish. That's the difference between a startup and a restaurant or a barber shop. You fry eggs or cut hair one customer at a time. Whereas if you solve a technical problem that a lot of people care about, you help everyone who uses your solution. That's leverage.

If you look at history, it seems that most people who got rich by creating wealth did it by developing new technology. You just can't fry eggs or cut hair fast enough. What made the Florentines rich in 1200 was the discovery of new techniques for making the high-tech product of the time, fine woven cloth. What made the Dutch rich in 1600 was the discovery of shipbuilding and navigation techniques that enabled them to dominate the seas of the Far East.

Fortunately there is a natural fit between smallness and solving hard problems. The leading edge of technology moves fast. Technology that's valuable today could be worthless in a couple years. Small companies are more at home in this world, because they don't have layers of bureaucracy to slow them down. Also, technical advances tend to come from unorthodox approaches, and small companies are less constrained by convention.

Big companies can develop technology. They just can't do it quickly. Their size makes them slow and prevents them from rewarding employees for the extraordinary effort required. So in practice big companies only get to develop technology in fields where large capital requirements prevent startups from competing with them, like microprocessors, power plants, or passenger aircraft. And even in those fields they depend heavily on startups for components and ideas.

It's obvious that biotech or software startups exist to solve hard technical problems, but I think it will also be found to be true in businesses that don't seem to be about technology. McDonald's, for example, grew big by designing a system, the McDonald's franchise, that could then be reproduced at will all over the face of the earth. A McDonald's franchise is controlled by rules so precise that it is practically a piece of software. Write once, run everywhere. Ditto for Wal-Mart. Sam Walton got rich not by being a retailer, but by designing a new kind of store.

Use difficulty as a guide not just in selecting the overall aim of your company, but also at decision points along the way. At Viaweb one of our rules of thumb was run upstairs. Suppose you are a little, nimble guy being chased by a big, fat, bully. You open a door and find yourself in a staircase. Do you go up or down? I say up. The bully can probably run downstairs as fast as you can. Going upstairs his bulk will be more of a disadvantage. Running upstairs is hard for you but even harder for him.

What this meant in practice was that we deliberately sought hard problems. If there were two features we could add to our software, both equally valuable in proportion to their difficulty, we'd always take the harder one. Not just because it was more valuable, but because it was harder. We delighted in forcing bigger, slower competitors to follow

us over difficult ground. Like guerillas, startups prefer the difficult terrain of the mountains, where the troops of the central government can't follow. I can remember times when we were just exhausted after wrestling all day with some horrible technical problem. And I'd be delighted, because something that was hard for us would be impossible for our competitors.

This is not just a good way to run a startup. It's what a startup is. Venture capitalists know about this and have a phrase for it: barriers to entry. If you go to a VC with a new idea and ask him to invest in it, one of the first things he'll ask is, how hard would this be for someone else to develop? That is, how much difficult ground have you put between yourself and potential pursuers? [7] And you had better have a convincing explanation of why your technology would be hard to duplicate. Otherwise as soon as some big company becomes aware of it, they'll make their own, and with their brand name, capital, and distribution clout, they'll take away your market overnight. You'd be like guerillas caught in the open field by regular army forces.

One way to put up barriers to entry is through patents. But patents may not provide much protection. Competitors commonly find ways to work around a patent. And if they can't, they may simply violate it and invite you to sue them. A big company is not afraid to be sued; it's an everyday thing for them. They'll make sure that suing them is expensive and takes a long time. Ever heard of Philo Farnsworth? He invented television. The reason you've never heard of him is that his company was not the one to make money from it. [8] The company that did was RCA, and Farnsworth's reward for his efforts was a decade of patent litigation.

Here, as so often, the best defense is a good offense. If you can develop technology that's simply too hard for competitors to duplicate, you don't need to rely on other defenses. Start by picking a hard problem, and then at every decision point, take the harder choice. [9]

The Catch(es)

If it were simply a matter of working harder than an ordinary employee and getting paid proportionately, it would obviously be a good deal to start a startup. Up to a point it would be more fun. I don't think many people like the slow pace of big companies, the interminable meetings, the water-cooler conversations, the clueless middle managers, and so on.

Unfortunately there are a couple catches. One is that you can't choose the point on the curve that you want to inhabit. You can't decide, for example, that you'd like to work just two or three times as hard, and get paid that much more. When you're running a startup, your competitors decide how hard you work. And they pretty much all make the same decision: as hard as you possibly can.

The other catch is that the payoff is only on average proportionate to your productivity. There is, as I said before, a large random multiplier in the success of any company. So in practice the deal is not that you're 30 times as productive and get paid 30 times as much. It is that you're 30 times as productive, and get paid between zero and a thousand times as much. If the mean is 30x, the median is probably zero. Most startups tank, and not just the dogfood portals we all heard about during the Internet Bubble. It's common for a startup to be developing a genuinely good product, take slightly too long to do it, run out of money, and have to shut down.

A startup is like a mosquito. A bear can absorb a hit and a crab is armored against one, but a mosquito is designed for one thing: to score. No energy is wasted on defense. The defense of mosquitos, as a species, is that there are a lot of them, but this is little consolation to the individual mosquito.

Startups, like mosquitos, tend to be an all-or-nothing proposition. And you don't generally know which of the two you're going to get till the last minute. Viaweb came close to tanking several times. Our trajectory was like a sine wave. Fortunately we got bought at the top of the cycle, but it was damned close. While we were visiting Yahoo in California to talk about selling the company to them, we had to borrow a conference room to reassure an investor who was about to back out of a new round of funding that we needed to stay alive.

The all-or-nothing aspect of startups was not something we wanted. Viaweb's hackers were all extremely risk-averse. If there had been some way just to work super hard and get paid for it, without having a lottery mixed in, we would have been delighted. We would have much preferred a 100% chance of $1 million to a 20% chance of $10 million, even though theoretically the second is worth twice as much. Unfortunately, there is not currently any space in the business world where you can get the first deal.

The closest you can get is by selling your startup in the early stages, giving up upside (and risk) for a smaller but guaranteed payoff. We had a chance to do this, and stupidly, as we then thought, let it slip by. After that we became comically eager to sell. For the next year or so, if anyone expressed the slightest curiosity about Viaweb we would try to sell them the company. But there were no takers, so we had to keep going.

It would have been a bargain to buy us at an early stage, but companies doing acquisitions are not looking for bargains. A company big enough to acquire startups will be big enough to be fairly conservative, and within the company the people in charge of acquisitions will be among the more conservative, because they are likely to be business school types who joined the company late. They would rather overpay for a safe choice. So it is easier to sell an established startup, even at a large premium, than an early-stage one.

Get Users

I think it's a good idea to get bought, if you can. Running a business is different from growing one. It is just as well to let a big company take over once you reach cruising altitude. It's also financially wiser, because selling allows you to diversify. What would you think of a financial advisor who put all his client's assets into one volatile stock?

How do you get bought? Mostly by doing the same things you'd do if you didn't intend to sell the company. Being profitable, for example. But getting bought is also an art in its own right, and one that we spent a lot of time trying to master.

Potential buyers will always delay if they can. The hard part about getting bought is getting them to act. For most people, the most powerful motivator is not the hope of gain, but the fear of loss. For potential acquirers, the most

powerful motivator is the prospect that one of their competitors will buy you. This, as we found, causes CEOs to take red-eyes. The second biggest is the worry that, if they don't buy you now, you'll continue to grow rapidly and will cost more to acquire later, or even become a competitor.

In both cases, what it all comes down to is users. You'd think that a company about to buy you would do a lot of research and decide for themselves how valuable your technology was. Not at all. What they go by is the number of users you have.

In effect, acquirers assume the customers know who has the best technology. And this is not as stupid as it sounds. Users are the only real proof that you've created wealth. Wealth is what people want, and if people aren't using your software, maybe it's not just because you're bad at marketing. Maybe it's because you haven't made what they want.

Venture capitalists have a list of danger signs to watch out for. Near the top is the company run by techno-weenies who are obsessed with solving interesting technical problems, instead of making users happy. In a startup, you're not just trying to solve problems. You're trying to solve problems that users care about.

So I think you should make users the test, just as acquirers do. Treat a startup as an optimization problem in which performance is measured by number of users. As anyone who has tried to optimize software knows, the key is measurement. When you try to guess where your program is slow, and what would make it faster, you almost always guess wrong.

Number of users may not be the perfect test, but it will be very close. It's what acquirers care about. It's what revenues depend on. It's what makes competitors unhappy. It's what impresses reporters, and potential new users. Certainly it's a better test than your a priori notions of what problems are important to solve, no matter how technically adept you are.

Among other things, treating a startup as an optimization problem will help you avoid another pitfall that VCs worry about, and rightly-- taking a long time to develop a product. Now we can recognize this as something hackers already know to avoid: premature optimization. Get a version 1.0 out there as soon as you can. Until you have some users to measure, you're optimizing based on guesses.

The ball you need to keep your eye on here is the underlying principle that wealth is what people want. If you plan to get rich by creating wealth, you have to know what people want. So few businesses really pay attention to making customers happy. How often do you walk into a store, or call a company on the phone, with a feeling of dread in the back of your mind? When you hear "your call is important to us, please stay on the line," do you think, oh good, now everything will be all right?

A restaurant can afford to serve the occasional burnt dinner. But in technology, you cook one thing and that's what everyone eats. So any difference between what people want and what you deliver is multiplied. You please or annoy customers wholesale. The closer you can get to what they want, the more wealth you generate.

Wealth and Power

Making wealth is not the only way to get rich. For most of human history it has not even been the most common. Until a few centuries ago, the main sources of wealth were mines, slaves and serfs, land, and cattle, and the only ways to acquire these rapidly were by inheritance, marriage, conquest, or confiscation. Naturally wealth had a bad reputation.

Two things changed. The first was the rule of law. For most of the world's history, if you did somehow accumulate a fortune, the ruler or his henchmen would find a way to steal it. But in medieval Europe something new happened. A new class of merchants and manufacturers began to collect in towns. [10] Together they were able to withstand the local feudal lord. So for the first time in our history, the bullies stopped stealing the nerds' lunch money. This was naturally a great incentive, and possibly indeed the main cause of the second big change, industrialization.

A great deal has been written about the causes of the Industrial Revolution. But surely a necessary, if not sufficient, condition was that people who made fortunes be able to enjoy them in peace. [11] One piece of evidence is what happened to countries that tried to return to the old model, like the Soviet Union, and to a lesser extent Britain under the labor governments of the 1960s and early 1970s. Take away the incentive of wealth, and technical innovation grinds to a halt.

Remember what a startup is, economically: a way of saying, I want to work faster. Instead of accumulating money slowly by being paid a regular wage for fifty years, I want to get it over with as soon as possible. So governments that forbid you to accumulate wealth are in effect decreeing that you work slowly. They're willing to let you earn $3 million over fifty years, but they're not willing to let you work so hard that you can do it in two. They are like the corporate boss that you can't go to and say, I want to work ten times as hard, so please pay me ten times a much. Except this is not a boss you can escape by starting your own company.

The problem with working slowly is not just that technical innovation happens slowly. It's that it tends not to happen at all. It's only when you're deliberately looking for hard problems, as a way to use speed to the greatest advantage, that you take on this kind of project. Developing new technology is a pain in the ass. It is, as Edison said, one percent inspiration and ninety-nine percent perspiration. Without the incentive of wealth, no one wants to do it. Engineers will work on sexy projects like fighter planes and moon rockets for ordinary salaries, but more mundane technologies like light bulbs or semiconductors have to be developed by entrepreneurs.

Startups are not just something that happened in Silicon Valley in the last couple decades. Since it became possible to get rich by creating wealth, everyone who has done it has used essentially the same recipe: measurement and leverage, where measurement comes from working with a small group, and leverage from developing new techniques. The recipe was the same in Florence in 1200 as it is in Santa Clara today.

Understanding this may help to answer an important question: why Europe grew so powerful. Was it something about the geography of Europe? Was it that Europeans are somehow racially superior? Was it their religion? The

answer (or at least the proximate cause) may be that the Europeans rode on the crest of a powerful new idea: allowing those who made a lot of money to keep it.

Once you're allowed to do that, people who want to get rich can do it by generating wealth instead of stealing it. The resulting technological growth translates not only into wealth but into military power. The theory that led to the stealth plane was developed by a Soviet mathematician. But because the Soviet Union didn't have a computer industry, it remained for them a theory; they didn't have hardware capable of executing the calculations fast enough to design an actual airplane.

In that respect the Cold War teaches the same lesson as World War II and, for that matter, most wars in recent history. Don't let a ruling class of warriors and politicians squash the entrepreneurs. The same recipe that makes individuals rich makes countries powerful. Let the nerds keep their lunch money, and you rule the world.

Notes

[1] One valuable thing you tend to get only in startups is uninterruptability. Different kinds of work have different time quanta. Someone proofreading a manuscript could probably be interrupted every fifteen minutes with little loss of productivity. But the time quantum for hacking is very long: it might take an hour just to load a problem into your head. So the cost of having someone from personnel call you about a form you forgot to fill out can be huge.

This is why hackers give you such a baleful stare as they turn from their screen to answer your question. Inside their heads a giant house of cards is tottering.

The mere possibility of being interrupted deters hackers from starting hard projects. This is why they tend to work late at night, and why it's next to impossible to write great software in a cubicle (except late at night).

One great advantage of startups is that they don't yet have any of the people who interrupt you. There is no personnel department, and thus no form nor anyone to call you about it.

[2] Faced with the idea that people working for startups might be 20 or 30 times as productive as those working for large companies, executives at large companies will naturally wonder, how could I get the people working for me to do that? The answer is simple: pay them to.

Internally most companies are run like Communist states. If you believe in free markets, why not turn your company into one?

Hypothesis: A company will be maximally profitable when each employee is paid in proportion to the wealth they generate.

[3] Until recently even governments sometimes didn't grasp the distinction between money and wealth. Adam Smith (Wealth of Nations, v:i) mentions several that tried to preserve their "wealth" by forbidding the export of gold or silver. But having more of the medium of exchange would not make a country richer; if you have more money chasing the same amount of material wealth, the only result is higher prices.

[4] There are many senses of the word "wealth," not all of them material. I'm not trying to make a deep philosophical point here about which is the true kind. I'm writing about one specific, rather technical sense of the word "wealth." What people will give you money for. This is an interesting sort of wealth to study, because it is the kind that prevents you from starving. And what people will give you money for depends on them, not you.

When you're starting a business, it's easy to slide into thinking that customers want what you do. During the Internet Bubble I talked to a woman who, because she liked the outdoors, was starting an "outdoor portal." You know what kind of business you should start if you like the outdoors? One to recover data from crashed hard disks.

What's the connection? None at all. Which is precisely my point. If you want to create wealth (in the narrow technical sense of not starving) then you should be especially skeptical about any plan that centers on things you like doing. That is where your idea of what's valuable is least likely to coincide with other people's.

[5] In the average car restoration you probably do make everyone else microscopically poorer, by doing a small amount of damage to the environment. While environmental costs should be taken into account, they don't make wealth a zero-sum game. For example, if you repair a machine that's broken because a part has come unscrewed, you create wealth with no environmental cost.

[5b] This essay was written before Firefox.

[6] Many people feel confused and depressed in their early twenties. Life seemed so much more fun in college. Well, of course it was. Don't be fooled by the surface similarities. You've gone from guest to servant. It's possible to have fun in this new world. Among other things, you now get to go behind the doors that say "authorized personnel only." But the change is a shock at first, and all the worse if you're not consciously aware of it.

[7] When VCs asked us how long it would take another startup to duplicate our software, we used to reply that they probably wouldn't be able to at all. I think this made us seem naive, or liars.

[8] Few technologies have one clear inventor. So as a rule, if you know the "inventor" of something (the telephone, the assembly line, the airplane, the light bulb, the transistor) it is because their company made money from it, and the company's PR people worked hard to spread the story. If you don't know who invented something (the

automobile, the television, the computer, the jet engine, the laser), it's because other companies made all the money.

[9] This is a good plan for life in general. If you have two choices, choose the harder. If you're trying to decide whether to go out running or sit home and watch TV, go running. Probably the reason this trick works so well is that when you have two choices and one is harder, the only reason you're even considering the other is laziness. You know in the back of your mind what's the right thing to do, and this trick merely forces you to acknowledge it.

[10] It is probably no accident that the middle class first appeared in northern Italy and the low countries, where there were no strong central governments. These two regions were the richest of their time and became the twin centers from which Renaissance civilization radiated. If they no longer play that role, it is because other places, like the United States, have been truer to the principles they discovered.

[11] It may indeed be a sufficient condition. But if so, why didn't the Industrial Revolution happen earlier? Two possible (and not incompatible) answers: (a) It did. The Industrial Revolution was one in a series. (b) Because in medieval towns, monopolies and guild regulations initially slowed the development of new means of production.

You Weren't Meant to Have a Boss

Want to start a startup? Get funded by Y Combinator.

March 2008, rev. June 2008

Technology tends to separate normal from natural. Our bodies weren't designed to eat the foods that people in rich countries eat, or to get so little exercise. There may be a similar problem with the way we work: a normal job may be as bad for us intellectually as white flour or sugar is for us physically.

I began to suspect this after spending several years working with startup founders. I've now worked with over 200 of them, and I've noticed a definite difference between programmers working on their own startups and those working for large organizations. I wouldn't say founders seem happier, necessarily; starting a startup can be very stressful. Maybe the best way to put it is to say that they're happier in the sense that your body is happier during a long run than sitting on a sofa eating doughnuts.

Though they're statistically abnormal, startup founders seem to be working in a way that's more natural for humans.

I was in Africa last year and saw a lot of animals in the wild that I'd only seen in zoos before. It was remarkable how different they seemed. Particularly lions. Lions in the wild seem about ten times more alive. They're like different animals. I suspect that working for oneself feels better to humans in much the same way that living in the wild must feel better to a wide-ranging predator like a lion. Life in a zoo is easier, but it isn't the life they were designed for.

Trees

What's so unnatural about working for a big company? The root of the problem is that humans weren't meant to work in such large groups.

Another thing you notice when you see animals in the wild is that each species thrives in groups of a certain size. A herd of impalas might have 100 adults; baboons maybe 20; lions rarely 10. Humans also seem designed to work in groups, and what I've read about hunter-gatherers accords with research on organizations and my own experience to suggest roughly what the ideal size is: groups of 8 work well; by 20 they're getting hard to manage; and a group of 50 is really unwieldy. [1]

Whatever the upper limit is, we are clearly not meant to work in groups of several hundred. And yet—for reasons having more to do with technology than human nature—a great many people work for companies with hundreds or thousands of employees.

Companies know groups that large wouldn't work, so they divide themselves into units small enough to work together. But to coordinate these they have to introduce something new: bosses.

These smaller groups are always arranged in a tree structure. Your boss is the point where your group attaches to the tree. But when you use this trick for dividing a large group into smaller ones, something strange happens that I've never heard anyone mention explicitly. In the group one level up from yours, your boss represents your entire group. A group of 10 managers is not merely a group of 10 people working together in the usual way. It's really a group of groups. Which means for a group of 10 managers to work together as if they were simply a group of 10 individuals, the group working for each manager would have to work as if they were a single person—the workers and manager would each share only one person's worth of freedom between them.

In practice a group of people are never able to act as if they were one person. But in a large organization divided into groups in this way, the pressure is always in that direction. Each group tries its best to work as if it were the small group of individuals that humans were designed to work in. That was the point of creating it. And when you propagate that constraint, the result is that each person gets freedom of action in inverse proportion to the size of the entire tree. [2]

Anyone who's worked for a large organization has felt this. You can feel the difference between working for a company with 100 employees and one with 10,000, even if your group has only 10 people.

Corn Syrup

A group of 10 people within a large organization is a kind of fake tribe. The number of people you interact with is about right. But something is missing: individual initiative. Tribes of hunter-gatherers have much more freedom. The leaders have a little more power than other members of the tribe, but they don't generally tell them what to do and when the way a boss can.

It's not your boss's fault. The real problem is that in the group above you in the hierarchy, your entire group is one virtual person. Your boss is just the way that constraint is imparted to you.

So working in a group of 10 people within a large organization feels both right and wrong at the same time. On the surface it feels like the kind of group you're meant to work in, but something major is missing. A job at a big company is like high fructose corn syrup: it has some of the qualities of things you're meant to like, but is disastrously lacking in others.

Indeed, food is an excellent metaphor to explain what's wrong with the usual sort of job.

For example, working for a big company is the default thing to do, at least for programmers. How bad could it be? Well, food shows that pretty clearly. If you were dropped at a random point in America today, nearly all the food around you would be bad for you. Humans were not designed to eat white flour, refined sugar, high fructose corn syrup, and hydrogenated vegetable oil. And yet if you analyzed the contents of the average grocery store you'd probably find these four ingredients accounted for most of the calories. "Normal" food is terribly bad for you. The only people who eat what humans were actually designed to eat are a few Birkenstock-wearing weirdos in Berkeley.

If "normal" food is so bad for us, why is it so common? There are two main reasons. One is that it has more immediate appeal. You may feel lousy an hour after eating that pizza, but eating the first couple bites feels great. The other is economies of scale. Producing junk food scales; producing fresh vegetables doesn't. Which means (a) junk food can be very cheap, and (b) it's worth spending a lot to market it.

If people have to choose between something that's cheap, heavily marketed, and appealing in the short term, and something that's expensive, obscure, and appealing in the long term, which do you think most will choose?

It's the same with work. The average MIT graduate wants to work at Google or Microsoft, because it's a recognized brand, it's safe, and they'll get paid a good salary right away. It's the job equivalent of the pizza they had for lunch. The drawbacks will only become apparent later, and then only in a vague sense of malaise.

And founders and early employees of startups, meanwhile, are like the Birkenstock-wearing weirdos of Berkeley: though a tiny minority of the population, they're the ones living as humans are meant to. In an artificial world, only extremists live naturally.

Programmers

The restrictiveness of big company jobs is particularly hard on programmers, because the essence of programming is to build new things. Sales people make much the same pitches every day; support people answer much the same questions; but once you've written a piece of code you don't need to write it again. So a programmer working as programmers are meant to is always making new things. And when you're part of an organization whose structure gives each person freedom in inverse proportion to the size of the tree, you're going to face resistance when you do something new.

This seems an inevitable consequence of bigness. It's true even in the smartest companies. I was talking recently to a founder who considered starting a startup right out of college, but went to work for Google instead because he thought he'd learn more there. He didn't learn as much as he expected. Programmers learn by doing, and most of the things he wanted to do, he couldn't—sometimes because the company wouldn't let him, but often because the company's code wouldn't let him. Between the drag of legacy code, the overhead of doing development in such a large organization, and the restrictions imposed by interfaces owned by other groups, he could only try a fraction of the things he would have liked to. He said he has learned much more in his own startup, despite the fact that he has

to do all the company's errands as well as programming, because at least when he's programming he can do whatever he wants.

An obstacle downstream propagates upstream. If you're not allowed to implement new ideas, you stop having them. And vice versa: when you can do whatever you want, you have more ideas about what to do. So working for yourself makes your brain more powerful in the same way a low-restriction exhaust system makes an engine more powerful.

Working for yourself doesn't have to mean starting a startup, of course. But a programmer deciding between a regular job at a big company and their own startup is probably going to learn more doing the startup.

You can adjust the amount of freedom you get by scaling the size of company you work for. If you start the company, you'll have the most freedom. If you become one of the first 10 employees you'll have almost as much freedom as the founders. Even a company with 100 people will feel different from one with 1000.

Working for a small company doesn't ensure freedom. The tree structure of large organizations sets an upper bound on freedom, not a lower bound. The head of a small company may still choose to be a tyrant. The point is that a large organization is compelled by its structure to be one.

Consequences

That has real consequences for both organizations and individuals. One is that companies will inevitably slow down as they grow larger, no matter how hard they try to keep their startup mojo. It's a consequence of the tree structure that every large organization is forced to adopt.

Or rather, a large organization could only avoid slowing down if they avoided tree structure. And since human nature limits the size of group that can work together, the only way I can imagine for larger groups to avoid tree structure would be to have no structure: to have each group actually be independent, and to work together the way components of a market economy do.

That might be worth exploring. I suspect there are already some highly partitionable businesses that lean this way. But I don't know any technology companies that have done it.

There is one thing companies can do short of structuring themselves as sponges: they can stay small. If I'm right, then it really pays to keep a company as small as it can be at every stage. Particularly a technology company. Which means it's doubly important to hire the best people. Mediocre hires hurt you twice: they get less done, but they also make you big, because you need more of them to solve a given problem.

For individuals the upshot is the same: aim small. It will always suck to work for large organizations, and the larger the organization, the more it will suck.

In an essay I wrote a couple years ago I advised graduating seniors to work for a couple years for another company before starting their own. I'd modify that now. Work for another company if you want to, but only for a small one, and if you want to start your own startup, go ahead.

The reason I suggested college graduates not start startups immediately was that I felt most would fail. And they will. But ambitious programmers are better off doing their own thing and failing than going to work at a big company. Certainly they'll learn more. They might even be better off financially. A lot of people in their early twenties get into debt, because their expenses grow even faster than the salary that seemed so high when they left school. At least if you start a startup and fail your net worth will be zero rather than negative. [3]

We've now funded so many different types of founders that we have enough data to see patterns, and there seems to be no benefit from working for a big company. The people who've worked for a few years do seem better than the ones straight out of college, but only because they're that much older.

The people who come to us from big companies often seem kind of conservative. It's hard to say how much is because big companies made them that way, and how much is the natural conservatism that made them work for the big companies in the first place. But certainly a large part of it is learned. I know because I've seen it burn off.

Having seen that happen so many times is one of the things that convinces me that working for oneself, or at least for a small group, is the natural way for programmers to live. Founders arriving at Y Combinator often have the downtrodden air of refugees. Three months later they're transformed: they have so much more confidence that they seem as if they've grown several inches taller. [4] Strange as this sounds, they seem both more worried and happier at the same time. Which is exactly how I'd describe the way lions seem in the wild.

Watching employees get transformed into founders makes it clear that the difference between the two is due mostly to environment—and in particular that the environment in big companies is toxic to programmers. In the first couple weeks of working on their own startup they seem to come to life, because finally they're working the way people are meant to.

Notes

[1] When I talk about humans being meant or designed to live a certain way, I mean by evolution.

[2] It's not only the leaves who suffer. The constraint propagates up as well as down. So managers are constrained too; instead of just doing things, they have to act through subordinates.

[3] Do not finance your startup with credit cards. Financing a startup with debt is usually a stupid move, and credit card debt stupidest of all. Credit card debt is a bad idea, period. It is a trap set by evil companies for the desperate and the foolish.

[4] The founders we fund used to be younger (initially we encouraged undergrads to apply), and the first couple times I saw this I used to wonder if they were actually getting physically taller.

Thanks to Trevor Blackwell, Ross Boucher, Aaron Iba, Abby Kirigin, Ivan Kirigin, Jessica Livingston, and Robert Morris for reading drafts of this.

Why to Not Not Start a Startup

Want to start a startup? Get funded by Y Combinator.

March 2007

(This essay is derived from talks at the 2007 Startup School and the Berkeley CSUA.)

We've now been doing Y Combinator long enough to have some data about success rates. Our first batch, in the summer of 2005, had eight startups in it. Of those eight, it now looks as if at least four succeeded. Three have been acquired: Reddit was a merger of two, Reddit and Infogami, and a third was acquired that we can't talk about yet. Another from that batch was Loopt, which is doing so well they could probably be acquired in about ten minutes if they wanted to.

So about half the founders from that first summer, less than two years ago, are now rich, at least by their standards. (One thing you learn when you get rich is that there are many degrees of it.)

I'm not ready to predict our success rate will stay as high as 50%. That first batch could have been an anomaly. But we should be able to do better than the oft-quoted (and probably made up) standard figure of 10%. I'd feel safe aiming at 25%.

Even the founders who fail don't seem to have such a bad time. Of those first eight startups, three are now probably dead. In two cases the founders just went on to do other things at the end of the summer. I don't think they were traumatized by the experience. The closest to a traumatic failure was Kiko, whose founders kept working on their startup for a whole year before being squashed by Google Calendar. But they ended up happy. They sold their software on eBay for a quarter of a million dollars. After they paid back their angel investors, they had about a year's salary each. [1] Then they immediately went on to start a new and much more exciting startup, Justin.TV.

So here is an even more striking statistic: 0% of that first batch had a terrible experience. They had ups and downs, like every startup, but I don't think any would have traded it for a job in a cubicle. And that statistic is probably not an anomaly. Whatever our long-term success rate ends up being, I think the rate of people who wish they'd gotten a regular job will stay close to 0%.

The big mystery to me is: why don't more people start startups? If nearly everyone who does it prefers it to a regular job, and a significant percentage get rich, why doesn't everyone want to do this? A lot of people think we get thousands of applications for each funding cycle. In fact we usually only get several hundred. Why don't more people apply? And while it must seem to anyone watching this world that startups are popping up like crazy, the number is

small compared to the number of people with the necessary skills. The great majority of programmers still go straight from college to cubicle, and stay there.

It seems like people are not acting in their own interest. What's going on? Well, I can answer that. Because of Y Combinator's position at the very start of the venture funding process, we're probably the world's leading experts on the psychology of people who aren't sure if they want to start a company.

There's nothing wrong with being unsure. If you're a hacker thinking about starting a startup and hesitating before taking the leap, you're part of a grand tradition. Larry and Sergey seem to have felt the same before they started Google, and so did Jerry and Filo before they started Yahoo. In fact, I'd guess the most successful startups are the ones started by uncertain hackers rather than gung-ho business guys.

We have some evidence to support this. Several of the most successful startups we've funded told us later that they only decided to apply at the last moment. Some decided only hours before the deadline.

The way to deal with uncertainty is to analyze it into components. Most people who are reluctant to do something have about eight different reasons mixed together in their heads, and don't know themselves which are biggest. Some will be justified and some bogus, but unless you know the relative proportion of each, you don't know whether your overall uncertainty is mostly justified or mostly bogus.

So I'm going to list all the components of people's reluctance to start startups, and explain which are real. Then would-be founders can use this as a checklist to examine their own feelings.

I admit my goal is to increase your self-confidence. But there are two things different here from the usual confidence-building exercise. One is that I'm motivated to be honest. Most people in the confidence-building business have already achieved their goal when you buy the book or pay to attend the seminar where they tell you how great you are. Whereas if I encourage people to start startups who shouldn't, I make my own life worse. If I encourage too many people to apply to Y Combinator, it just means more work for me, because I have to read all the applications.

The other thing that's going to be different is my approach. Instead of being positive, I'm going to be negative. Instead of telling you "come on, you can do it" I'm going to consider all the reasons you aren't doing it, and show why most (but not all) should be ignored. We'll start with the one everyone's born with.

1. Too young

A lot of people think they're too young to start a startup. Many are right. The median age worldwide is about 27, so probably a third of the population can truthfully say they're too young.

What's too young? One of our goals with Y Combinator was to discover the lower bound on the age of startup founders. It always seemed to us that investors were too conservative here—that they wanted to fund professors, when really they should be funding grad students or even undergrads.

The main thing we've discovered from pushing the edge of this envelope is not where the edge is, but how fuzzy it is. The outer limit may be as low as 16. We don't look beyond 18 because people younger than that can't legally enter into contracts. But the most successful founder we've funded so far, Sam Altman, was 19 at the time.

Sam Altman, however, is an outlying data point. When he was 19, he seemed like he had a 40 year old inside him. There are other 19 year olds who are 12 inside.

There's a reason we have a distinct word "adult" for people over a certain age. There is a threshold you cross. It's conventionally fixed at 21, but different people cross it at greatly varying ages. You're old enough to start a startup if you've crossed this threshold, whatever your age.

How do you tell? There are a couple tests adults use. I realized these tests existed after meeting Sam Altman, actually. I noticed that I felt like I was talking to someone much older. Afterward I wondered, what am I even measuring? What made him seem older?

One test adults use is whether you still have the kid flake reflex. When you're a little kid and you're asked to do something hard, you can cry and say "I can't do it" and the adults will probably let you off. As a kid there's a magic button you can press by saying "I'm just a kid" that will get you out of most difficult situations. Whereas adults, by definition, are not allowed to flake. They still do, of course, but when they do they're ruthlessly pruned.

The other way to tell an adult is by how they react to a challenge. Someone who's not yet an adult will tend to respond to a challenge from an adult in a way that acknowledges their dominance. If an adult says "that's a stupid idea," a kid will either crawl away with his tail between his legs, or rebel. But rebelling presumes inferiority as much as submission. The adult response to "that's a stupid idea," is simply to look the other person in the eye and say "Really? Why do you think so?"

There are a lot of adults who still react childishly to challenges, of course. What you don't often find are kids who react to challenges like adults. When you do, you've found an adult, whatever their age.

2. Too inexperienced

I once wrote that startup founders should be at least 23, and that people should work for another company for a few years before starting their own. I no longer believe that, and what changed my mind is the example of the startups we've funded.

I still think 23 is a better age than 21. But the best way to get experience if you're 21 is to start a startup. So, paradoxically, if you're too inexperienced to start a startup, what you should do is start one. That's a way more efficient cure for inexperience than a normal job. In fact, getting a normal job may actually make you less able to start a startup, by turning you into a tame animal who thinks he needs an office to work in and a product manager to tell him what software to write.

What really convinced me of this was the Kikos. They started a startup right out of college. Their inexperience caused them to make a lot of mistakes. But by the time we funded their second startup, a year later, they had become extremely formidable. They were certainly not tame animals. And there is no way they'd have grown so much if they'd spent that year working at Microsoft, or even Google. They'd still have been diffident junior programmers.

So now I'd advise people to go ahead and start startups right out of college. There's no better time to take risks than when you're young. Sure, you'll probably fail. But even failure will get you to the ultimate goal faster than getting a job.

It worries me a bit to be saying this, because in effect we're advising people to educate themselves by failing at our expense, but it's the truth.

3. Not determined enough

You need a lot of determination to succeed as a startup founder. It's probably the single best predictor of success.

Some people may not be determined enough to make it. It's hard for me to say for sure, because I'm so determined that I can't imagine what's going on in the heads of people who aren't. But I know they exist.

Most hackers probably underestimate their determination. I've seen a lot become visibly more determined as they get used to running a startup. I can think of several we've funded who would have been delighted at first to be bought for $2 million, but are now set on world domination.

How can you tell if you're determined enough, when Larry and Sergey themselves were unsure at first about starting a company? I'm guessing here, but I'd say the test is whether you're sufficiently driven to work on your own projects. Though they may have been unsure whether they wanted to start a company, it doesn't seem as if Larry and Sergey were meek little research assistants, obediently doing their advisors' bidding. They started projects of their own.

4. Not smart enough

You may need to be moderately smart to succeed as a startup founder. But if you're worried about this, you're probably mistaken. If you're smart enough to worry that you might not be smart enough to start a startup, you probably are.

And in any case, starting a startup just doesn't require that much intelligence. Some startups do. You have to be good at math to write Mathematica. But most companies do more mundane stuff where the decisive factor is effort, not brains. Silicon Valley can warp your perspective on this, because there's a cult of smartness here. People who aren't smart at least try to act that way. But if you think it takes a lot of intelligence to get rich, try spending a couple days in some of the fancier bits of New York or LA.

If you don't think you're smart enough to start a startup doing something technically difficult, just write enterprise software. Enterprise software companies aren't technology companies, they're sales companies, and sales depends mostly on effort.

5. Know nothing about business

This is another variable whose coefficient should be zero. You don't need to know anything about business to start a startup. The initial focus should be the product. All you need to know in this phase is how to build things people want. If you succeed, you'll have to think about how to make money from it. But this is so easy you can pick it up on the fly.

I get a fair amount of flak for telling founders just to make something great and not worry too much about making money. And yet all the empirical evidence points that way: pretty much 100% of startups that make something popular manage to make money from it. And acquirers tell me privately that revenue is not what they buy startups for, but their strategic value. Which means, because they made something people want. Acquirers know the rule holds for them too: if users love you, you can always make money from that somehow, and if they don't, the cleverest business model in the world won't save you.

So why do so many people argue with me? I think one reason is that they hate the idea that a bunch of twenty year olds could get rich from building something cool that doesn't make any money. They just don't want that to be possible. But how possible it is doesn't depend on how much they want it to be.

For a while it annoyed me to hear myself described as some kind of irresponsible pied piper, leading impressionable young hackers down the road to ruin. But now I realize this kind of controversy is a sign of a good idea.

The most valuable truths are the ones most people don't believe. They're like undervalued stocks. If you start with them, you'll have the whole field to yourself. So when you find an idea you know is good but most people disagree with, you should not merely ignore their objections, but push aggressively in that direction. In this case, that means you should seek out ideas that would be popular but seem hard to make money from.

We'll bet a seed round you can't make something popular that we can't figure out how to make money from.

6. No cofounder

Not having a cofounder is a real problem. A startup is too much for one person to bear. And though we differ from other investors on a lot of questions, we all agree on this. All investors, without exception, are more likely to fund you with a cofounder than without.

We've funded two single founders, but in both cases we suggested their first priority should be to find a cofounder. Both did. But we'd have preferred them to have cofounders before they applied. It's not super hard to get a cofounder for a project that's just been funded, and we'd rather have cofounders committed enough to sign up for something super hard.

If you don't have a cofounder, what should you do? Get one. It's more important than anything else. If there's no one where you live who wants to start a startup with you, move where there are people who do. If no one wants to work with you on your current idea, switch to an idea people want to work on.

If you're still in school, you're surrounded by potential cofounders. A few years out it gets harder to find them. Not only do you have a smaller pool to draw from, but most already have jobs, and perhaps even families to support. So if you had friends in college you used to scheme about startups with, stay in touch with them as well as you can. That may help keep the dream alive.

It's possible you could meet a cofounder through something like a user's group or a conference. But I wouldn't be too optimistic. You need to work with someone to know whether you want them as a cofounder. [2]

The real lesson to draw from this is not how to find a cofounder, but that you should start startups when you're young and there are lots of them around.

7. No idea

In a sense, it's not a problem if you don't have a good idea, because most startups change their idea anyway. In the average Y Combinator startup, I'd guess 70% of the idea is new at the end of the first three months. Sometimes it's 100%.

In fact, we're so sure the founders are more important than the initial idea that we're going to try something new this funding cycle. We're going to let people apply with no idea at all. If you want, you can answer the question on the application form that asks what you're going to do with "We have no idea." If you seem really good we'll accept you anyway. We're confident we can sit down with you and cook up some promising project.

Really this just codifies what we do already. We put little weight on the idea. We ask mainly out of politeness. The kind of question on the application form that we really care about is the one where we ask what cool things you've

made. If what you've made is version one of a promising startup, so much the better, but the main thing we care about is whether you're good at making things. Being lead developer of a popular open source project counts almost as much.

That solves the problem if you get funded by Y Combinator. What about in the general case? Because in another sense, it is a problem if you don't have an idea. If you start a startup with no idea, what do you do next?

So here's the brief recipe for getting startup ideas. Find something that's missing in your own life, and supply that need—no matter how specific to you it seems. Steve Wozniak built himself a computer; who knew so many other people would want them? A need that's narrow but genuine is a better starting point than one that's broad but hypothetical. So even if the problem is simply that you don't have a date on Saturday night, if you can think of a way to fix that by writing software, you're onto something, because a lot of other people have the same problem.

8. No room for more startups

A lot of people look at the ever-increasing number of startups and think "this can't continue." Implicit in their thinking is a fallacy: that there is some limit on the number of startups there could be. But this is false. No one claims there's any limit on the number of people who can work for salary at 1000-person companies. Why should there be any limit on the number who can work for equity at 5-person companies? [3]

Nearly everyone who works is satisfying some kind of need. Breaking up companies into smaller units doesn't make those needs go away. Existing needs would probably get satisfied more efficiently by a network of startups than by a few giant, hierarchical organizations, but I don't think that would mean less opportunity, because satisfying current needs would lead to more. Certainly this tends to be the case in individuals. Nor is there anything wrong with that. We take for granted things that medieval kings would have considered effeminate luxuries, like whole buildings heated to spring temperatures year round. And if things go well, our descendants will take for granted things we would consider shockingly luxurious. There is no absolute standard for material wealth. Health care is a component of it, and that alone is a black hole. For the foreseeable future, people will want ever more material wealth, so there is no limit to the amount of work available for companies, and for startups in particular.

Usually the limited-room fallacy is not expressed directly. Usually it's implicit in statements like "there are only so many startups Google, Microsoft, and Yahoo can buy." Maybe, though the list of acquirers is a lot longer than that. And whatever you think of other acquirers, Google is not stupid. The reason big companies buy startups is that they've created something valuable. And why should there be any limit to the number of valuable startups companies can acquire, any more than there is a limit to the amount of wealth individual people want? Maybe there would be practical limits on the number of startups any one acquirer could assimilate, but if there is value to be had, in the form of upside that founders are willing to forgo in return for an immediate payment, acquirers will evolve to consume it. Markets are pretty smart that way.

9. Family to support

This one is real. I wouldn't advise anyone with a family to start a startup. I'm not saying it's a bad idea, just that I don't want to take responsibility for advising it. I'm willing to take responsibility for telling 22 year olds to start startups. So what if they fail? They'll learn a lot, and that job at Microsoft will still be waiting for them if they need it. But I'm not prepared to cross moms.

What you can do, if you have a family and want to start a startup, is start a consulting business you can then gradually turn into a product business. Empirically the chances of pulling that off seem very small. You're never going to produce Google this way. But at least you'll never be without an income.

Another way to decrease the risk is to join an existing startup instead of starting your own. Being one of the first employees of a startup is a lot like being a founder, in both the good ways and the bad. You'll be roughly 1/n^2 founder, where n is your employee number.

As with the question of cofounders, the real lesson here is to start startups when you're young.

10. Independently wealthy

This is my excuse for not starting a startup. Startups are stressful. Why do it if you don't need the money? For every "serial entrepreneur," there are probably twenty sane ones who think "Start another company? Are you crazy?"

I've come close to starting new startups a couple times, but I always pull back because I don't want four years of my life to be consumed by random schleps. I know this business well enough to know you can't do it half-heartedly. What makes a good startup founder so dangerous is his willingness to endure infinite schleps.

There is a bit of a problem with retirement, though. Like a lot of people, I like to work. And one of the many weird little problems you discover when you get rich is that a lot of the interesting people you'd like to work with are not rich. They need to work at something that pays the bills. Which means if you want to have them as colleagues, you have to work at something that pays the bills too, even though you don't need to. I think this is what drives a lot of serial entrepreneurs, actually.

That's why I love working on Y Combinator so much. It's an excuse to work on something interesting with people I like.

11. Not ready for commitment

This was my reason for not starting a startup for most of my twenties. Like a lot of people that age, I valued freedom most of all. I was reluctant to do anything that required a commitment of more than a few months. Nor would I have wanted to do anything that completely took over my life the way a startup does. And that's fine. If you want to spend

your time travelling around, or playing in a band, or whatever, that's a perfectly legitimate reason not to start a company.

If you start a startup that succeeds, it's going to consume at least three or four years. (If it fails, you'll be done a lot quicker.) So you shouldn't do it if you're not ready for commitments on that scale. Be aware, though, that if you get a regular job, you'll probably end up working there for as long as a startup would take, and you'll find you have much less spare time than you might expect. So if you're ready to clip on that ID badge and go to that orientation session, you may also be ready to start that startup.

12. Need for structure

I'm told there are people who need structure in their lives. This seems to be a nice way of saying they need someone to tell them what to do. I believe such people exist. There's plenty of empirical evidence: armies, religious cults, and so on. They may even be the majority.

If you're one of these people, you probably shouldn't start a startup. In fact, you probably shouldn't even go to work for one. In a good startup, you don't get told what to do very much. There may be one person whose job title is CEO, but till the company has about twelve people no one should be telling anyone what to do. That's too inefficient. Each person should just do what they need to without anyone telling them.

If that sounds like a recipe for chaos, think about a soccer team. Eleven people manage to work together in quite complicated ways, and yet only in occasional emergencies does anyone tell anyone else what to do. A reporter once asked David Beckham if there were any language problems at Real Madrid, since the players were from about eight different countries. He said it was never an issue, because everyone was so good they never had to talk. They all just did the right thing.

How do you tell if you're independent-minded enough to start a startup? If you'd bristle at the suggestion that you aren't, then you probably are.

13. Fear of uncertainty

Perhaps some people are deterred from starting startups because they don't like the uncertainty. If you go to work for Microsoft, you can predict fairly accurately what the next few years will be like—all too accurately, in fact. If you start a startup, anything might happen.

Well, if you're troubled by uncertainty, I can solve that problem for you: if you start a startup, it will probably fail. Seriously, though, this is not a bad way to think about the whole experience. Hope for the best, but expect the worst. In the worst case, it will at least be interesting. In the best case you might get rich.

No one will blame you if the startup tanks, so long as you made a serious effort. There may once have been a time when employers would regard that as a mark against you, but they wouldn't now. I asked managers at big companies, and they all said they'd prefer to hire someone who'd tried to start a startup and failed over someone who'd spent the same time working at a big company.

Nor will investors hold it against you, as long as you didn't fail out of laziness or incurable stupidity. I'm told there's a lot of stigma attached to failing in other places—in Europe, for example. Not here. In America, companies, like practically everything else, are disposable.

14. Don't realize what you're avoiding

One reason people who've been out in the world for a year or two make better founders than people straight from college is that they know what they're avoiding. If their startup fails, they'll have to get a job, and they know how much jobs suck.

If you've had summer jobs in college, you may think you know what jobs are like, but you probably don't. Summer jobs at technology companies are not real jobs. If you get a summer job as a waiter, that's a real job. Then you have to carry your weight. But software companies don't hire students for the summer as a source of cheap labor. They do it in the hope of recruiting them when they graduate. So while they're happy if you produce, they don't expect you to.

That will change if you get a real job after you graduate. Then you'll have to earn your keep. And since most of what big companies do is boring, you're going to have to work on boring stuff. Easy, compared to college, but boring. At first it may seem cool to get paid for doing easy stuff, after paying to do hard stuff in college. But that wears off after a few months. Eventually it gets demoralizing to work on dumb stuff, even if it's easy and you get paid a lot.

And that's not the worst of it. The thing that really sucks about having a regular job is the expectation that you're supposed to be there at certain times. Even Google is afflicted with this, apparently. And what this means, as everyone who's had a regular job can tell you, is that there are going to be times when you have absolutely no desire to work on anything, and you're going to have to go to work anyway and sit in front of your screen and pretend to. To someone who likes work, as most good hackers do, this is torture.

In a startup, you skip all that. There's no concept of office hours in most startups. Work and life just get mixed together. But the good thing about that is that no one minds if you have a life at work. In a startup you can do whatever you want most of the time. If you're a founder, what you want to do most of the time is work. But you never have to pretend to.

If you took a nap in your office in a big company, it would seem unprofessional. But if you're starting a startup and you fall asleep in the middle of the day, your cofounders will just assume you were tired.

15. Parents want you to be a doctor

A significant number of would-be startup founders are probably dissuaded from doing it by their parents. I'm not going to say you shouldn't listen to them. Families are entitled to their own traditions, and who am I to argue with them? But I will give you a couple reasons why a safe career might not be what your parents really want for you.

One is that parents tend to be more conservative for their kids than they would be for themselves. This is actually a rational response to their situation. Parents end up sharing more of their kids' ill fortune than good fortune. Most parents don't mind this; it's part of the job; but it does tend to make them excessively conservative. And erring on the side of conservatism is still erring. In almost everything, reward is proportionate to risk. So by protecting their kids from risk, parents are, without realizing it, also protecting them from rewards. If they saw that, they'd want you to take more risks.

The other reason parents may be mistaken is that, like generals, they're always fighting the last war. If they want you to be a doctor, odds are it's not just because they want you to help the sick, but also because it's a prestigious and lucrative career. [4] But not so lucrative or prestigious as it was when their opinions were formed. When I was a kid in the seventies, a doctor was the thing to be. There was a sort of golden triangle involving doctors, Mercedes 450SLs, and tennis. All three vertices now seem pretty dated.

The parents who want you to be a doctor may simply not realize how much things have changed. Would they be that unhappy if you were Steve Jobs instead? So I think the way to deal with your parents' opinions about what you should do is to treat them like feature requests. Even if your only goal is to please them, the way to do that is not simply to give them what they ask for. Instead think about why they're asking for something, and see if there's a better way to give them what they need.

16. A job is the default

This leads us to the last and probably most powerful reason people get regular jobs: it's the default thing to do. Defaults are enormously powerful, precisely because they operate without any conscious choice.

To almost everyone except criminals, it seems an axiom that if you need money, you should get a job. Actually this tradition is not much more than a hundred years old. Before that, the default way to make a living was by farming. It's a bad plan to treat something only a hundred years old as an axiom. By historical standards, that's something that's changing pretty rapidly.

We may be seeing another such change right now. I've read a lot of economic history, and I understand the startup world pretty well, and it now seems to me fairly likely that we're seeing the beginning of a change like the one from farming to manufacturing.

And you know what? If you'd been around when that change began (around 1000 in Europe) it would have seemed to nearly everyone that running off to the city to make your fortune was a crazy thing to do. Though serfs were in principle forbidden to leave their manors, it can't have been that hard to run away to a city. There were no guards patrolling the perimeter of the village. What prevented most serfs from leaving was that it seemed insanely risky. Leave one's plot of land? Leave the people you'd spent your whole life with, to live in a giant city of three or four thousand complete strangers? How would you live? How would you get food, if you didn't grow it?

Frightening as it seemed to them, it's now the default with us to live by our wits. So if it seems risky to you to start a startup, think how risky it once seemed to your ancestors to live as we do now. Oddly enough, the people who know this best are the very ones trying to get you to stick to the old model. How can Larry and Sergey say you should come work as their employee, when they didn't get jobs themselves?

Now we look back on medieval peasants and wonder how they stood it. How grim it must have been to till the same fields your whole life with no hope of anything better, under the thumb of lords and priests you had to give all your surplus to and acknowledge as your masters. I wouldn't be surprised if one day people look back on what we consider a normal job in the same way. How grim it would be to commute every day to a cubicle in some soulless office complex, and be told what to do by someone you had to acknowledge as a boss—someone who could call you into their office and say "take a seat," and you'd sit! Imagine having to ask permission to release software to users. Imagine being sad on Sunday afternoons because the weekend was almost over, and tomorrow you'd have to get up and go to work. How did they stand it?

It's exciting to think we may be on the cusp of another shift like the one from farming to manufacturing. That's why I care about startups. Startups aren't interesting just because they're a way to make a lot of money. I couldn't care less about other ways to do that, like speculating in securities. At most those are interesting the way puzzles are. There's more going on with startups. They may represent one of those rare, historic shifts in the way wealth is created.

That's ultimately what drives us to work on Y Combinator. We want to make money, if only so we don't have to stop doing it, but that's not the main goal. There have only been a handful of these great economic shifts in human history. It would be an amazing hack to make one happen faster.

Notes

[1] The only people who lost were us. The angels had convertible debt, so they had first claim on the proceeds of the auction. Y Combinator only got 38 cents on the dollar.

[2] The best kind of organization for that might be an open source project, but those don't involve a lot of face to face meetings. Maybe it would be worth starting one that did.

[3] There need to be some number of big companies to acquire the startups, so the number of big companies couldn't decrease to zero.

[4] Thought experiment: If doctors did the same work, but as impoverished outcasts, which parents would still want their kids to be doctors?

Why to Start a Startup in a Bad Economy

Want to start a startup? Get funded by Y Combinator.

October 2008

The economic situation is apparently so grim that some experts fear we may be in for a stretch as bad as the mid seventies.

When Microsoft and Apple were founded.

As those examples suggest, a recession may not be such a bad time to start a startup. I'm not claiming it's a particularly good time either. The truth is more boring: the state of the economy doesn't matter much either way.

If we've learned one thing from funding so many startups, it's that they succeed or fail based on the qualities of the founders. The economy has some effect, certainly, but as a predictor of success it's rounding error compared to the founders.

Which means that what matters is who you are, not when you do it. If you're the right sort of person, you'll win even in a bad economy. And if you're not, a good economy won't save you. Someone who thinks "I better not start a startup now, because the economy is so bad" is making the same mistake as the people who thought during the Bubble "all I have to do is start a startup, and I'll be rich."

So if you want to improve your chances, you should think far more about who you can recruit as a cofounder than the state of the economy. And if you're worried about threats to the survival of your company, don't look for them in the news. Look in the mirror.

But for any given team of founders, would it not pay to wait till the economy is better before taking the leap? If you're starting a restaurant, maybe, but not if you're working on technology. Technology progresses more or less independently of the stock market. So for any given idea, the payoff for acting fast in a bad economy will be higher than for waiting. Microsoft's first product was a Basic interpreter for the Altair. That was exactly what the world needed in 1975, but if Gates and Allen had decided to wait a few years, it would have been too late.

Of course, the idea you have now won't be the last you have. There are always new ideas. But if you have a specific idea you want to act on, act now.

That doesn't mean you can ignore the economy. Both customers and investors will be feeling pinched. It's not necessarily a problem if customers feel pinched: you may even be able to benefit from it, by making things that save money. Startups often make things cheaper, so in that respect they're better positioned to prosper in a recession than big companies.

Investors are more of a problem. Startups generally need to raise some amount of external funding, and investors tend to be less willing to invest in bad times. They shouldn't be. Everyone knows you're supposed to buy when times are bad and sell when times are good. But of course what makes investing so counterintuitive is that in equity markets, good times are defined as everyone thinking it's time to buy. You have to be a contrarian to be correct, and by definition only a minority of investors can be.

So just as investors in 1999 were tripping over one another trying to buy into lousy startups, investors in 2009 will presumably be reluctant to invest even in good ones.

You'll have to adapt to this. But that's nothing new: startups always have to adapt to the whims of investors. Ask any founder in any economy if they'd describe investors as fickle, and watch the face they make. Last year you had to be prepared to explain how your startup was viral. Next year you'll have to explain how it's recession-proof.

(Those are both good things to be. The mistake investors make is not the criteria they use but that they always tend to focus on one to the exclusion of the rest.)

Fortunately the way to make a startup recession-proof is to do exactly what you should do anyway: run it as cheaply as possible. For years I've been telling founders that the surest route to success is to be the cockroaches of the corporate world. The immediate cause of death in a startup is always running out of money. So the cheaper your company is to operate, the harder it is to kill. And fortunately it has gotten very cheap to run a startup. A recession will if anything make it cheaper still.

If nuclear winter really is here, it may be safer to be a cockroach even than to keep your job. Customers may drop off individually if they can no longer afford you, but you're not going to lose them all at once; markets don't "reduce headcount."

What if you quit your job to start a startup that fails, and you can't find another? That could be a problem if you work in sales or marketing. In those fields it can take months to find a new job in a bad economy. But hackers seem to be more liquid. Good hackers can always get some kind of job. It might not be your dream job, but you're not going to starve.

Another advantage of bad times is that there's less competition. Technology trains leave the station at regular intervals. If everyone else is cowering in a corner, you may have a whole car to yourself.

You're an investor too. As a founder, you're buying stock with work: the reason Larry and Sergey are so rich is not so much that they've done work worth tens of billions of dollars, but that they were the first investors in Google. And like any investor you should buy when times are bad.

Were you nodding in agreement, thinking "stupid investors" a few paragraphs ago when I was talking about how investors are reluctant to put money into startups in bad markets, even though that's the time they should rationally be most willing to buy? Well, founders aren't much better. When times get bad, hackers go to grad school. And no doubt that will happen this time too. In fact, what makes the preceding paragraph true is that most readers won't believe it—at least to the extent of acting on it.

So maybe a recession is a good time to start a startup. It's hard to say whether advantages like lack of competition outweigh disadvantages like reluctant investors. But it doesn't matter much either way. It's the people that matter. And for a given set of people working on a given technology, the time to act is always now.

Ideas for Startups

Want to start a startup? Get funded by Y Combinator.

October 2005

(This essay is derived from a talk at the 2005 Startup School.)

How do you get good ideas for startups? That's probably the number one question people ask me.

I'd like to reply with another question: why do people think it's hard to come up with ideas for startups?

That might seem a stupid thing to ask. Why do they think it's hard? If people can't do it, then it is hard, at least for them. Right?

Well, maybe not. What people usually say is not that they can't think of ideas, but that they don't have any. That's not quite the same thing. It could be the reason they don't have any is that they haven't tried to generate them.

I think this is often the case. I think people believe that coming up with ideas for startups is very hard-- that it must be very hard-- and so they don't try do to it. They assume ideas are like miracles: they either pop into your head or they don't.

I also have a theory about why people think this. They overvalue ideas. They think creating a startup is just a matter of implementing some fabulous initial idea. And since a successful startup is worth millions of dollars, a good idea is therefore a million dollar idea.

If coming up with an idea for a startup equals coming up with a million dollar idea, then of course it's going to seem hard. Too hard to bother trying. Our instincts tell us something so valuable would not be just lying around for anyone to discover.

Actually, startup ideas are not million dollar ideas, and here's an experiment you can try to prove it: just try to sell one. Nothing evolves faster than markets. The fact that there's no market for startup ideas suggests there's no demand. Which means, in the narrow sense of the word, that startup ideas are worthless.

Questions

The fact is, most startups end up nothing like the initial idea. It would be closer to the truth to say the main value of your initial idea is that, in the process of discovering it's broken, you'll come up with your real idea.

The initial idea is just a starting point-- not a blueprint, but a question. It might help if they were expressed that way. Instead of saying that your idea is to make a collaborative, web-based spreadsheet, say: could one make a collaborative, web-based spreadsheet? A few grammatical tweaks, and a woefully incomplete idea becomes a promising question to explore.

There's a real difference, because an assertion provokes objections in a way a question doesn't. If you say: I'm going to build a web-based spreadsheet, then critics-- the most dangerous of which are in your own head-- will immediately reply that you'd be competing with Microsoft, that you couldn't give people the kind of UI they expect, that users wouldn't want to have their data on your servers, and so on.

A question doesn't seem so challenging. It becomes: let's try making a web-based spreadsheet and see how far we get. And everyone knows that if you tried this you'd be able to make something useful. Maybe what you'd end up with wouldn't even be a spreadsheet. Maybe it would be some kind of new spreasheet-like collaboration tool that doesn't even have a name yet. You wouldn't have thought of something like that except by implementing your way toward it.

Treating a startup idea as a question changes what you're looking for. If an idea is a blueprint, it has to be right. But if it's a question, it can be wrong, so long as it's wrong in a way that leads to more ideas.

One valuable way for an idea to be wrong is to be only a partial solution. When someone's working on a problem that seems too big, I always ask: is there some way to bite off some subset of the problem, then gradually expand from there? That will generally work unless you get trapped on a local maximum, like 1980s-style AI, or C.

Upwind

So far, we've reduced the problem from thinking of a million dollar idea to thinking of a mistaken question. That doesn't seem so hard, does it?

To generate such questions you need two things: to be familiar with promising new technologies, and to have the right kind of friends. New technologies are the ingredients startup ideas are made of, and conversations with friends are the kitchen they're cooked in.

Universities have both, and that's why so many startups grow out of them. They're filled with new technologies, because they're trying to produce research, and only things that are new count as research. And they're full of

exactly the right kind of people to have ideas with: the other students, who will be not only smart but elastic-minded to a fault.

The opposite extreme would be a well-paying but boring job at a big company. Big companies are biased against new technologies, and the people you'd meet there would be wrong too.

In an essay I wrote for high school students, I said a good rule of thumb was to stay upwind-- to work on things that maximize your future options. The principle applies for adults too, though perhaps it has to be modified to: stay upwind for as long as you can, then cash in the potential energy you've accumulated when you need to pay for kids.

I don't think people consciously realize this, but one reason downwind jobs like churning out Java for a bank pay so well is precisely that they are downwind. The market price for that kind of work is higher because it gives you fewer options for the future. A job that lets you work on exciting new stuff will tend to pay less, because part of the compensation is in the form of the new skills you'll learn.

Grad school is the other end of the spectrum from a coding job at a big company: the pay's low but you spend most of your time working on new stuff. And of course, it's called "school," which makes that clear to everyone, though in fact all jobs are some percentage school.

The right environment for having startup ideas need not be a university per se. It just has to be a situation with a large percentage of school.

It's obvious why you want exposure to new technology, but why do you need other people? Can't you just think of new ideas yourself? The empirical answer is: no. Even Einstein needed people to bounce ideas off. Ideas get developed in the process of explaining them to the right kind of person. You need that resistance, just as a carver needs the resistance of the wood.

This is one reason Y Combinator has a rule against investing in startups with only one founder. Practically every successful company has at least two. And because startup founders work under great pressure, it's critical they be friends.

I didn't realize it till I was writing this, but that may help explain why there are so few female startup founders. I read on the Internet (so it must be true) that only 1.7% of VC-backed startups are founded by women. The percentage of female hackers is small, but not that small. So why the discrepancy?

When you realize that successful startups tend to have multiple founders who were already friends, a possible explanation emerges. People's best friends are likely to be of the same sex, and if one group is a minority in some population, pairs of them will be a minority squared. [1]

Doodling

What these groups of co-founders do together is more complicated than just sitting down and trying to think of ideas. I suspect the most productive setup is a kind of together-alone-together sandwich. Together you talk about some hard problem, probably getting nowhere. Then, the next morning, one of you has an idea in the shower about how to solve it. He runs eagerly to to tell the others, and together they work out the kinks.

What happens in that shower? It seems to me that ideas just pop into my head. But can we say more than that?

Taking a shower is like a form of meditation. You're alert, but there's nothing to distract you. It's in a situation like this, where your mind is free to roam, that it bumps into new ideas.

What happens when your mind wanders? It may be like doodling. Most people have characteristic ways of doodling. This habit is unconscious, but not random: I found my doodles changed after I started studying painting. I started to make the kind of gestures I'd make if I were drawing from life. They were atoms of drawing, but arranged randomly. [2]

Perhaps letting your mind wander is like doodling with ideas. You have certain mental gestures you've learned in your work, and when you're not paying attention, you keep making these same gestures, but somewhat randomly. In effect, you call the same functions on random arguments. That's what a metaphor is: a function applied to an argument of the wrong type.

Conveniently, as I was writing this, my mind wandered: would it be useful to have metaphors in a programming language? I don't know; I don't have time to think about this. But it's convenient because this is an example of what I mean by habits of mind. I spend a lot of time thinking about language design, and my habit of always asking "would x be useful in a programming language" just got invoked.

If new ideas arise like doodles, this would explain why you have to work at something for a while before you have any. It's not just that you can't judge ideas till you're an expert in a field. You won't even generate ideas, because you won't have any habits of mind to invoke.

Of course the habits of mind you invoke on some field don't have to be derived from working in that field. In fact, it's often better if they're not. You're not just looking for good ideas, but for good new ideas, and you have a better chance of generating those if you combine stuff from distant fields. As hackers, one of our habits of mind is to ask, could one open-source x? For example, what if you made an open-source operating system? A fine idea, but not very novel. Whereas if you ask, could you make an open-source play? you might be onto something.

Are some kinds of work better sources of habits of mind than others? I suspect harder fields may be better sources, because to attack hard problems you need powerful solvents. I find math is a good source of metaphors-- good enough that it's worth studying just for that. Related fields are also good sources, especially when they're related in

unexpected ways. Everyone knows computer science and electrical engineering are related, but precisely because everyone knows it, importing ideas from one to the other doesn't yield great profits. It's like importing something from Wisconsin to Michigan. Whereas (I claim) hacking and painting are also related, in the sense that hackers and painters are both makers, and this source of new ideas is practically virgin territory.

Problems

In theory you could stick together ideas at random and see what you came up with. What if you built a peer-to-peer dating site? Would it be useful to have an automatic book? Could you turn theorems into a commodity? When you assemble ideas at random like this, they may not be just stupid, but semantically ill-formed. What would it even mean to make theorems a commodity? You got me. I didn't think of that idea, just its name.

You might come up with something useful this way, but I never have. It's like knowing a fabulous sculpture is hidden inside a block of marble, and all you have to do is remove the marble that isn't part of it. It's an encouraging thought, because it reminds you there is an answer, but it's not much use in practice because the search space is too big.

I find that to have good ideas I need to be working on some problem. You can't start with randomness. You have to start with a problem, then let your mind wander just far enough for new ideas to form.

In a way, it's harder to see problems than their solutions. Most people prefer to remain in denial about problems. It's obvious why: problems are irritating. They're problems! Imagine if people in 1700 saw their lives the way we'd see them. It would have been unbearable. This denial is such a powerful force that, even when presented with possible solutions, people often prefer to believe they wouldn't work.

I saw this phenomenon when I worked on spam filters. In 2002, most people preferred to ignore spam, and most of those who didn't preferred to believe the heuristic filters then available were the best you could do.

I found spam intolerable, and I felt it had to be possible to recognize it statistically. And it turns out that was all you needed to solve the problem. The algorithm I used was ridiculously simple. Anyone who'd really tried to solve the problem would have found it. It was just that no one had really tried to solve the problem. [3]

Let me repeat that recipe: finding the problem intolerable and feeling it must be possible to solve it. Simple as it seems, that's the recipe for a lot of startup ideas.

Wealth

So far most of what I've said applies to ideas in general. What's special about startup ideas? Startup ideas are ideas for companies, and companies have to make money. And the way to make money is to make something people want.

Wealth is what people want. I don't mean that as some kind of philosophical statement; I mean it as a tautology.

So an idea for a startup is an idea for something people want. Wouldn't any good idea be something people want? Unfortunately not. I think new theorems are a fine thing to create, but there is no great demand for them. Whereas there appears to be great demand for celebrity gossip magazines. Wealth is defined democratically. Good ideas and valuable ideas are not quite the same thing; the difference is individual tastes.

But valuable ideas are very close to good ideas, especially in technology. I think they're so close that you can get away with working as if the goal were to discover good ideas, so long as, in the final stage, you stop and ask: will people actually pay for this? Only a few ideas are likely to make it that far and then get shot down; RPN calculators might be one example.

One way to make something people want is to look at stuff people use now that's broken. Dating sites are a prime example. They have millions of users, so they must be promising something people want. And yet they work horribly. Just ask anyone who uses them. It's as if they used the worse-is-better approach but stopped after the first stage and handed the thing over to marketers.

Of course, the most obvious breakage in the average computer user's life is Windows itself. But this is a special case: you can't defeat a monopoly by a frontal attack. Windows can and will be overthrown, but not by giving people a better desktop OS. The way to kill it is to redefine the problem as a superset of the current one. The problem is not, what operating system should people use on desktop computers? but how should people use applications? There are answers to that question that don't even involve desktop computers.

Everyone thinks Google is going to solve this problem, but it is a very subtle one, so subtle that a company as big as Google might well get it wrong. I think the odds are better than 50-50 that the Windows killer-- or more accurately, Windows transcender-- will come from some little startup.

Another classic way to make something people want is to take a luxury and make it into a commmodity. People must want something if they pay a lot for it. And it is a very rare product that can't be made dramatically cheaper if you try.

This was Henry Ford's plan. He made cars, which had been a luxury item, into a commodity. But the idea is much older than Henry Ford. Water mills transformed mechanical power from a luxury into a commodity, and they were used in the Roman empire. Arguably pastoralism transformed a luxury into a commodity.

When you make something cheaper you can sell more of them. But if you make something dramatically cheaper you often get qualitative changes, because people start to use it in different ways. For example, once computers get so cheap that most people can have one of their own, you can use them as communication devices.

Often to make something dramatically cheaper you have to redefine the problem. The Model T didn't have all the features previous cars did. It only came in black, for example. But it solved the problem people cared most about, which was getting from place to place.

One of the most useful mental habits I know I learned from Michael Rabin: that the best way to solve a problem is often to redefine it. A lot of people use this technique without being consciously aware of it, but Rabin was spectacularly explicit. You need a big prime number? Those are pretty expensive. How about if I give you a big number that only has a 10 to the minus 100 chance of not being prime? Would that do? Well, probably; I mean, that's probably smaller than the chance that I'm imagining all this anyway.

Redefining the problem is a particularly juicy heuristic when you have competitors, because it's so hard for rigid-minded people to follow. You can work in plain sight and they don't realize the danger. Don't worry about us. We're just working on search. Do one thing and do it well, that's our motto.

Making things cheaper is actually a subset of a more general technique: making things easier. For a long time it was most of making things easier, but now that the things we build are so complicated, there's another rapidly growing subset: making things easier to use.

This is an area where there's great room for improvement. What you want to be able to say about technology is: it just works. How often do you say that now?

Simplicity takes effort-- genius, even. The average programmer seems to produce UI designs that are almost willfully bad. I was trying to use the stove at my mother's house a couple weeks ago. It was a new one, and instead of physical knobs it had buttons and an LED display. I tried pressing some buttons I thought would cause it to get hot, and you know what it said? "Err." Not even "Error." "Err." You can't just say "Err" to the user of a stove. You should design the UI so that errors are impossible. And the boneheads who designed this stove even had an example of such a UI to work from: the old one. You turn one knob to set the temperature and another to set the timer. What was wrong with that? It just worked.

It seems that, for the average engineer, more options just means more rope to hang yourself. So if you want to start a startup, you can take almost any existing technology produced by a big company, and assume you could build something way easier to use.

Design for Exit

Success for a startup approximately equals getting bought. You need some kind of exit strategy, because you can't get the smartest people to work for you without giving them options likely to be worth something. Which means you either have to get bought or go public, and the number of startups that go public is very small.

If success probably means getting bought, should you make that a conscious goal? The old answer was no: you were supposed to pretend that you wanted to create a giant, public company, and act surprised when someone made you an offer. Really, you want to buy us? Well, I suppose we'd consider it, for the right price.

I think things are changing. If 98% of the time success means getting bought, why not be open about it? If 98% of the time you're doing product development on spec for some big company, why not think of that as your task? One advantage of this approach is that it gives you another source of ideas: look at big companies, think what they should be doing, and do it yourself. Even if they already know it, you'll probably be done faster.

Just be sure to make something multiple acquirers will want. Don't fix Windows, because the only potential acquirer is Microsoft, and when there's only one acquirer, they don't have to hurry. They can take their time and copy you instead of buying you. If you want to get market price, work on something where there's competition.

If an increasing number of startups are created to do product development on spec, it will be a natural counterweight to monopolies. Once some type of technology is captured by a monopoly, it will only evolve at big company rates instead of startup rates, whereas alternatives will evolve with especial speed. A free market interprets monopoly as damage and routes around it.

The Woz Route

The most productive way to generate startup ideas is also the most unlikely-sounding: by accident. If you look at how famous startups got started, a lot of them weren't initially supposed to be startups. Lotus began with a program Mitch Kapor wrote for a friend. Apple got started because Steve Wozniak wanted to build microcomputers, and his employer, Hewlett-Packard, wouldn't let him do it at work. Yahoo began as David Filo's personal collection of links.

This is not the only way to start startups. You can sit down and consciously come up with an idea for a company; we did. But measured in total market cap, the build-stuff-for-yourself model might be more fruitful. It certainly has to be the most fun way to come up with startup ideas. And since a startup ought to have multiple founders who were already friends before they decided to start a company, the rather surprising conclusion is that the best way to generate startup ideas is to do what hackers do for fun: cook up amusing hacks with your friends.

It seems like it violates some kind of conservation law, but there it is: the best way to get a "million dollar idea" is just to do what hackers enjoy doing anyway.

Notes

[1] This phenomenon may account for a number of discrepancies currently blamed on various forbidden isms. Never attribute to malice what can be explained by math.

[2] A lot of classic abstract expressionism is doodling of this type: artists trained to paint from life using the same gestures but without using them to represent anything. This explains why such paintings are (slightly) more interesting than random marks would be.

[3] Bill Yerazunis had solved the problem, but he got there by another path. He made a general-purpose file classifier so good that it also worked for spam.

Relentlessly Resourceful

Want to start a startup? Get funded by Y Combinator.

March 2009

A couple days ago I finally got being a good startup founder down to two words: relentlessly resourceful.

Till then the best I'd managed was to get the opposite quality down to one: hapless. Most dictionaries say hapless means unlucky. But the dictionaries are not doing a very good job. A team that outplays its opponents but loses because of a bad decision by the referee could be called unlucky, but not hapless. Hapless implies passivity. To be hapless is to be battered by circumstances—to let the world have its way with you, instead of having your way with the world. [1]

Unfortunately there's no antonym of hapless, which makes it difficult to tell founders what to aim for. "Don't be hapless" is not much of rallying cry.

It's not hard to express the quality we're looking for in metaphors. The best is probably a running back. A good running back is not merely determined, but flexible as well. They want to get downfield, but they adapt their plans on the fly.

Unfortunately this is just a metaphor, and not a useful one to most people outside the US. "Be like a running back" is no better than "Don't be hapless."

But finally I've figured out how to express this quality directly. I was writing a talk for investors, and I had to explain what to look for in founders. What would someone who was the opposite of hapless be like? They'd be relentlessly resourceful. Not merely relentless. That's not enough to make things go your way except in a few mostly uninteresting domains. In any interesting domain, the difficulties will be novel. Which means you can't simply plow through them, because you don't know initially how hard they are; you don't know whether you're about to plow through a block of foam or granite. So you have to be resourceful. You have to keep trying new things.

Be relentlessly resourceful.

That sounds right, but is it simply a description of how to be successful in general? I don't think so. This isn't the recipe for success in writing or painting, for example. In that kind of work the recipe is more to be actively curious.

Resourceful implies the obstacles are external, which they generally are in startups. But in writing and painting they're mostly internal; the obstacle is your own obtuseness. [2]

There probably are other fields where "relentlessly resourceful" is the recipe for success. But though other fields may share it, I think this is the best short description we'll find of what makes a good startup founder. I doubt it could be made more precise.

Now that we know what we're looking for, that leads to other questions. For example, can this quality be taught? After four years of trying to teach it to people, I'd say that yes, surprisingly often it can. Not to everyone, but to many people. [3] Some people are just constitutionally passive, but others have a latent ability to be relentlessly resourceful that only needs to be brought out.

This is particularly true of young people who have till now always been under the thumb of some kind of authority. Being relentlessly resourceful is definitely not the recipe for success in big companies, or in most schools. I don't even want to think what the recipe is in big companies, but it is certainly longer and messier, involving some combination of resourcefulness, obedience, and building alliances.

Identifying this quality also brings us closer to answering a question people often wonder about: how many startups there could be. There is not, as some people seem to think, any economic upper bound on this number. There's no reason to believe there is any limit on the amount of newly created wealth consumers can absorb, any more than there is a limit on the number of theorems that can be proven. So probably the limiting factor on the number of startups is the pool of potential founders. Some people would make good founders, and others wouldn't. And now that we can say what makes a good founder, we know how to put an upper bound on the size of the pool.

This test is also useful to individuals. If you want to know whether you're the right sort of person to start a startup, ask yourself whether you're relentlessly resourceful. And if you want to know whether to recruit someone as a cofounder, ask if they are.

You can even use it tactically. If I were running a startup, this would be the phrase I'd tape to the mirror. "Make something people want" is the destination, but "Be relentlessly resourceful" is how you get there.

The 18 Mistakes That Kill Startups

Want to start a startup? Get funded by Y Combinator.

October 2006

In the Q & A period after a recent talk, someone asked what made startups fail. After standing there gaping for a few seconds I realized this was kind of a trick question. It's equivalent to asking how to make a startup succeed—if you avoid every cause of failure, you succeed—and that's too big a question to answer on the fly.

Afterwards I realized it could be helpful to look at the problem from this direction. If you have a list of all the things you shouldn't do, you can turn that into a recipe for succeeding just by negating. And this form of list may be more useful in practice. It's easier to catch yourself doing something you shouldn't than always to remember to do something you should. [1]

In a sense there's just one mistake that kills startups: not making something users want. If you make something users want, you'll probably be fine, whatever else you do or don't do. And if you don't make something users want, then you're dead, whatever else you do or don't do. So really this is a list of 18 things that cause startups not to make something users want. Nearly all failure funnels through that.

1. Single Founder

Have you ever noticed how few successful startups were founded by just one person? Even companies you think of as having one founder, like Oracle, usually turn out to have more. It seems unlikely this is a coincidence.

What's wrong with having one founder? To start with, it's a vote of no confidence. It probably means the founder couldn't talk any of his friends into starting the company with him. That's pretty alarming, because his friends are the ones who know him best.

But even if the founder's friends were all wrong and the company is a good bet, he's still at a disadvantage. Starting a startup is too hard for one person. Even if you could do all the work yourself, you need colleagues to brainstorm with, to talk you out of stupid decisions, and to cheer you up when things go wrong.

The last one might be the most important. The low points in a startup are so low that few could bear them alone. When you have multiple founders, esprit de corps binds them together in a way that seems to violate conservation

laws. Each thinks "I can't let my friends down." This is one of the most powerful forces in human nature, and it's missing when there's just one founder.

2. Bad Location

Startups prosper in some places and not others. Silicon Valley dominates, then Boston, then Seattle, Austin, Denver, and New York. After that there's not much. Even in New York the number of startups per capita is probably a 20th of what it is in Silicon Valley. In towns like Houston and Chicago and Detroit it's too small to measure.

Why is the falloff so sharp? Probably for the same reason it is in other industries. What's the sixth largest fashion center in the US? The sixth largest center for oil, or finance, or publishing? Whatever they are they're probably so far from the top that it would be misleading even to call them centers.

It's an interesting question why cities become startup hubs, but the reason startups prosper in them is probably the same as it is for any industry: that's where the experts are. Standards are higher; people are more sympathetic to what you're doing; the kind of people you want to hire want to live there; supporting industries are there; the people you run into in chance meetings are in the same business. Who knows exactly how these factors combine to boost startups in Silicon Valley and squish them in Detroit, but it's clear they do from the number of startups per capita in each.

3. Marginal Niche

Most of the groups that apply to Y Combinator suffer from a common problem: choosing a small, obscure niche in the hope of avoiding competition.

If you watch little kids playing sports, you notice that below a certain age they're afraid of the ball. When the ball comes near them their instinct is to avoid it. I didn't make a lot of catches as an eight year old outfielder, because whenever a fly ball came my way, I used to close my eyes and hold my glove up more for protection than in the hope of catching it.

Choosing a marginal project is the startup equivalent of my eight year old strategy for dealing with fly balls. If you make anything good, you're going to have competitors, so you may as well face that. You can only avoid competition by avoiding good ideas.

I think this shrinking from big problems is mostly unconscious. It's not that people think of grand ideas but decide to pursue smaller ones because they seem safer. Your unconscious won't even let you think of grand ideas. So the solution may be to think about ideas without involving yourself. What would be a great idea for someone else to do as a startup?

4. Derivative Idea

Many of the applications we get are imitations of some existing company. That's one source of ideas, but not the best. If you look at the origins of successful startups, few were started in imitation of some other startup. Where did they get their ideas? Usually from some specific, unsolved problem the founders identified.

Our startup made software for making online stores. When we started it, there wasn't any; the few sites you could order from were hand-made at great expense by web consultants. We knew that if online shopping ever took off, these sites would have to be generated by software, so we wrote some. Pretty straightforward.

It seems like the best problems to solve are ones that affect you personally. Apple happened because Steve Wozniak wanted a computer, Google because Larry and Sergey couldn't find stuff online, Hotmail because Sabeer Bhatia and Jack Smith couldn't exchange email at work.

So instead of copying the Facebook, with some variation that the Facebook rightly ignored, look for ideas from the other direction. Instead of starting from companies and working back to the problems they solved, look for problems and imagine the company that might solve them. [2] What do people complain about? What do you wish there was?

5. Obstinacy

In some fields the way to succeed is to have a vision of what you want to achieve, and to hold true to it no matter what setbacks you encounter. Starting startups is not one of them. The stick-to-your-vision approach works for something like winning an Olympic gold medal, where the problem is well-defined. Startups are more like science, where you need to follow the trail wherever it leads.

So don't get too attached to your original plan, because it's probably wrong. Most successful startups end up doing something different than they originally intended—often so different that it doesn't even seem like the same company. You have to be prepared to see the better idea when it arrives. And the hardest part of that is often discarding your old idea.

But openness to new ideas has to be tuned just right. Switching to a new idea every week will be equally fatal. Is there some kind of external test you can use? One is to ask whether the ideas represent some kind of progression. If in each new idea you're able to re-use most of what you built for the previous ones, then you're probably in a process that converges. Whereas if you keep restarting from scratch, that's a bad sign.

Fortunately there's someone you can ask for advice: your users. If you're thinking about turning in some new direction and your users seem excited about it, it's probably a good bet.

6. Hiring Bad Programmers

I forgot to include this in the early versions of the list, because nearly all the founders I know are programmers. This is not a serious problem for them. They might accidentally hire someone bad, but it's not going to kill the company. In a pinch they can do whatever's required themselves.

But when I think about what killed most of the startups in the e-commerce business back in the 90s, it was bad programmers. A lot of those companies were started by business guys who thought the way startups worked was that you had some clever idea and then hired programmers to implement it. That's actually much harder than it sounds—almost impossibly hard in fact—because business guys can't tell which are the good programmers. They don't even get a shot at the best ones, because no one really good wants a job implementing the vision of a business guy.

In practice what happens is that the business guys choose people they think are good programmers (it says here on his resume that he's a Microsoft Certified Developer) but who aren't. Then they're mystified to find that their startup lumbers along like a World War II bomber while their competitors scream past like jet fighters. This kind of startup is in the same position as a big company, but without the advantages.

So how do you pick good programmers if you're not a programmer? I don't think there's an answer. I was about to say you'd have to find a good programmer to help you hire people. But if you can't recognize good programmers, how would you even do that?

7. Choosing the Wrong Platform

A related problem (since it tends to be done by bad programmers) is choosing the wrong platform. For example, I think a lot of startups during the Bubble killed themselves by deciding to build server-based applications on Windows. Hotmail was still running on FreeBSD for years after Microsoft bought it, presumably because Windows couldn't handle the load. If Hotmail's founders had chosen to use Windows, they would have been swamped.

PayPal only just dodged this bullet. After they merged with X.com, the new CEO wanted to switch to Windows—even after PayPal cofounder Max Levchin showed that their software scaled only 1% as well on Windows as Unix. Fortunately for PayPal they switched CEOs instead.

Platform is a vague word. It could mean an operating system, or a programming language, or a "framework" built on top of a programming language. It implies something that both supports and limits, like the foundation of a house.

The scary thing about platforms is that there are always some that seem to outsiders to be fine, responsible choices and yet, like Windows in the 90s, will destroy you if you choose them. Java applets were probably the most spectacular example. This was supposed to be the new way of delivering applications. Presumably it killed just about 100% of the startups who believed that.

How do you pick the right platforms? The usual way is to hire good programmers and let them choose. But there is a trick you could use if you're not a programmer: visit a top computer science department and see what they use in research projects.

8. Slowness in Launching

Companies of all sizes have a hard time getting software done. It's intrinsic to the medium; software is always 85% done. It takes an effort of will to push through this and get something released to users. [3]

Startups make all kinds of excuses for delaying their launch. Most are equivalent to the ones people use for procrastinating in everyday life. There's something that needs to happen first. Maybe. But if the software were 100% finished and ready to launch at the push of a button, would they still be waiting?

One reason to launch quickly is that it forces you to actually finish some quantum of work. Nothing is truly finished till it's released; you can see that from the rush of work that's always involved in releasing anything, no matter how finished you thought it was. The other reason you need to launch is that it's only by bouncing your idea off users that you fully understand it.

Several distinct problems manifest themselves as delays in launching: working too slowly; not truly understanding the problem; fear of having to deal with users; fear of being judged; working on too many different things; excessive perfectionism. Fortunately you can combat all of them by the simple expedient of forcing yourself to launch something fairly quickly.

9. Launching Too Early

Launching too slowly has probably killed a hundred times more startups than launching too fast, but it is possible to launch too fast. The danger here is that you ruin your reputation. You launch something, the early adopters try it out, and if it's no good they may never come back.

So what's the minimum you need to launch? We suggest startups think about what they plan to do, identify a core that's both (a) useful on its own and (b) something that can be incrementally expanded into the whole project, and then get that done as soon as possible.

This is the same approach I (and many other programmers) use for writing software. Think about the overall goal, then start by writing the smallest subset of it that does anything useful. If it's a subset, you'll have to write it anyway, so in the worst case you won't be wasting your time. But more likely you'll find that implementing a working subset is both good for morale and helps you see more clearly what the rest should do.

The early adopters you need to impress are fairly tolerant. They don't expect a newly launched product to do everything; it just has to do something.

10. Having No Specific User in Mind

You can't build things users like without understanding them. I mentioned earlier that the most successful startups seem to have begun by trying to solve a problem their founders had. Perhaps there's a rule here: perhaps you create wealth in proportion to how well you understand the problem you're solving, and the problems you understand best are your own. [4]

That's just a theory. What's not a theory is the converse: if you're trying to solve problems you don't understand, you're hosed.

And yet a surprising number of founders seem willing to assume that someone, they're not sure exactly who, will want what they're building. Do the founders want it? No, they're not the target market. Who is? Teenagers. People interested in local events (that one is a perennial tarpit). Or "business" users. What business users? Gas stations? Movie studios? Defense contractors?

You can of course build something for users other than yourself. We did. But you should realize you're stepping into dangerous territory. You're flying on instruments, in effect, so you should (a) consciously shift gears, instead of assuming you can rely on your intuitions as you ordinarily would, and (b) look at the instruments.

In this case the instruments are the users. When designing for other people you have to be empirical. You can no longer guess what will work; you have to find users and measure their responses. So if you're going to make something for teenagers or "business" users or some other group that doesn't include you, you have to be able to talk some specific ones into using what you're making. If you can't, you're on the wrong track.

11. Raising Too Little Money

Most successful startups take funding at some point. Like having more than one founder, it seems a good bet statistically. How much should you take, though?

Startup funding is measured in time. Every startup that isn't profitable (meaning nearly all of them, initially) has a certain amount of time left before the money runs out and they have to stop. This is sometimes referred to as runway, as in "How much runway do you have left?" It's a good metaphor because it reminds you that when the money runs out you're going to be airborne or dead.

Too little money means not enough to get airborne. What airborne means depends on the situation. Usually you have to advance to a visibly higher level: if all you have is an idea, a working prototype; if you have a prototype,

launching; if you're launched, significant growth. It depends on investors, because until you're profitable that's who you have to convince.

So if you take money from investors, you have to take enough to get to the next step, whatever that is. [5] Fortunately you have some control over both how much you spend and what the next step is. We advise startups to set both low, initially: spend practically nothing, and make your initial goal simply to build a solid prototype. This gives you maximum flexibility.

12. Spending Too Much

It's hard to distinguish spending too much from raising too little. If you run out of money, you could say either was the cause. The only way to decide which to call it is by comparison with other startups. If you raised five million and ran out of money, you probably spent too much.

Burning through too much money is not as common as it used to be. Founders seem to have learned that lesson. Plus it keeps getting cheaper to start a startup. So as of this writing few startups spend too much. None of the ones we've funded have. (And not just because we make small investments; many have gone on to raise further rounds.)

The classic way to burn through cash is by hiring a lot of people. This bites you twice: in addition to increasing your costs, it slows you down—so money that's getting consumed faster has to last longer. Most hackers understand why that happens; Fred Brooks explained it in The Mythical Man-Month.

We have three general suggestions about hiring: (a) don't do it if you can avoid it, (b) pay people with equity rather than salary, not just to save money, but because you want the kind of people who are committed enough to prefer that, and (c) only hire people who are either going to write code or go out and get users, because those are the only things you need at first.

13. Raising Too Much Money

It's obvious how too little money could kill you, but is there such a thing as having too much?

Yes and no. The problem is not so much the money itself as what comes with it. As one VC who spoke at Y Combinator said, "Once you take several million dollars of my money, the clock is ticking." If VCs fund you, they're not going to let you just put the money in the bank and keep operating as two guys living on ramen. They want that money to go to work. [6] At the very least you'll move into proper office space and hire more people. That will change the atmosphere, and not entirely for the better. Now most of your people will be employees rather than founders. They won't be as committed; they'll need to be told what to do; they'll start to engage in office politics.

When you raise a lot of money, your company moves to the suburbs and has kids.

Perhaps more dangerously, once you take a lot of money it gets harder to change direction. Suppose your initial plan was to sell something to companies. After taking VC money you hire a sales force to do that. What happens now if you realize you should be making this for consumers instead of businesses? That's a completely different kind of selling. What happens, in practice, is that you don't realize that. The more people you have, the more you stay pointed in the same direction.

Another drawback of large investments is the time they take. The time required to raise money grows with the amount. [7] When the amount rises into the millions, investors get very cautious. VCs never quite say yes or no; they just engage you in an apparently endless conversation. Raising VC scale investments is thus a huge time sink—more work, probably, than the startup itself. And you don't want to be spending all your time talking to investors while your competitors are spending theirs building things.

We advise founders who go on to seek VC money to take the first reasonable deal they get. If you get an offer from a reputable firm at a reasonable valuation with no unusually onerous terms, just take it and get on with building the company. [8] Who cares if you could get a 30% better deal elsewhere? Economically, startups are an all-or-nothing game. Bargain-hunting among investors is a waste of time.

14. Poor Investor Management

As a founder, you have to manage your investors. You shouldn't ignore them, because they may have useful insights. But neither should you let them run the company. That's supposed to be your job. If investors had sufficient vision to run the companies they fund, why didn't they start them?

Pissing off investors by ignoring them is probably less dangerous than caving in to them. In our startup, we erred on the ignoring side. A lot of our energy got drained away in disputes with investors instead of going into the product. But this was less costly than giving in, which would probably have destroyed the company. If the founders know what they're doing, it's better to have half their attention focused on the product than the full attention of investors who don't.

How hard you have to work on managing investors usually depends on how much money you've taken. When you raise VC-scale money, the investors get a great deal of control. If they have a board majority, they're literally your bosses. In the more common case, where founders and investors are equally represented and the deciding vote is cast by neutral outside directors, all the investors have to do is convince the outside directors and they control the company.

If things go well, this shouldn't matter. So long as you seem to be advancing rapidly, most investors will leave you alone. But things don't always go smoothly in startups. Investors have made trouble even for the most successful companies. One of the most famous examples is Apple, whose board made a nearly fatal blunder in firing Steve Jobs. Apparently even Google got a lot of grief from their investors early on.

15. Sacrificing Users to (Supposed) Profit

When I said at the beginning that if you make something users want, you'll be fine, you may have noticed I didn't mention anything about having the right business model. That's not because making money is unimportant. I'm not suggesting that founders start companies with no chance of making money in the hope of unloading them before they tank. The reason we tell founders not to worry about the business model initially is that making something people want is so much harder.

I don't know why it's so hard to make something people want. It seems like it should be straightforward. But you can tell it must be hard by how few startups do it.

Because making something people want is so much harder than making money from it, you should leave business models for later, just as you'd leave some trivial but messy feature for version 2. In version 1, solve the core problem. And the core problem in a startup is how to create wealth (= how much people want something x the number who want it), not how to convert that wealth into money.

The companies that win are the ones that put users first. Google, for example. They made search work, then worried about how to make money from it. And yet some startup founders still think it's irresponsible not to focus on the business model from the beginning. They're often encouraged in this by investors whose experience comes from less malleable industries.

It is irresponsible not to think about business models. It's just ten times more irresponsible not to think about the product.

16. Not Wanting to Get Your Hands Dirty

Nearly all programmers would rather spend their time writing code and have someone else handle the messy business of extracting money from it. And not just the lazy ones. Larry and Sergey apparently felt this way too at first. After developing their new search algorithm, the first thing they tried was to get some other company to buy it.

Start a company? Yech. Most hackers would rather just have ideas. But as Larry and Sergey found, there's not much of a market for ideas. No one trusts an idea till you embody it in a product and use that to grow a user base. Then they'll pay big time.

Maybe this will change, but I doubt it will change much. There's nothing like users for convincing acquirers. It's not just that the risk is decreased. The acquirers are human, and they have a hard time paying a bunch of young guys millions of dollars just for being clever. When the idea is embodied in a company with a lot of users, they can tell themselves they're buying the users rather than the cleverness, and this is easier for them to swallow. [9]

If you're going to attract users, you'll probably have to get up from your computer and go find some. It's unpleasant work, but if you can make yourself do it you have a much greater chance of succeeding. In the first batch of startups we funded, in the summer of 2005, most of the founders spent all their time building their applications. But there was one who was away half the time talking to executives at cell phone companies, trying to arrange deals. Can you imagine anything more painful for a hacker? [10] But it paid off, because this startup seems the most successful of that group by an order of magnitude.

If you want to start a startup, you have to face the fact that you can't just hack. At least one hacker will have to spend some of the time doing business stuff.

17. Fights Between Founders

Fights between founders are surprisingly common. About 20% of the startups we've funded have had a founder leave. It happens so often that we've reversed our attitude to vesting. We still don't require it, but now we advise founders to vest so there will be an orderly way for people to quit.

A founder leaving doesn't necessarily kill a startup, though. Plenty of successful startups have had that happen. [11] Fortunately it's usually the least committed founder who leaves. If there are three founders and one who was lukewarm leaves, big deal. If you have two and one leaves, or a guy with critical technical skills leaves, that's more of a problem. But even that is survivable. Blogger got down to one person, and they bounced back.

Most of the disputes I've seen between founders could have been avoided if they'd been more careful about who they started a company with. Most disputes are not due to the situation but the people. Which means they're inevitable. And most founders who've been burned by such disputes probably had misgivings, which they suppressed, when they started the company. Don't suppress misgivings. It's much easier to fix problems before the company is started than after. So don't include your housemate in your startup because he'd feel left out otherwise. Don't start a company with someone you dislike because they have some skill you need and you worry you won't find anyone else. The people are the most important ingredient in a startup, so don't compromise there.

18. A Half-Hearted Effort

The failed startups you hear most about are the spectactular flameouts. Those are actually the elite of failures. The most common type is not the one that makes spectacular mistakes, but the one that doesn't do much of anything—the one we never even hear about, because it was some project a couple guys started on the side while working on their day jobs, but which never got anywhere and was gradually abandoned.

Statistically, if you want to avoid failure, it would seem like the most important thing is to quit your day job. Most founders of failed startups don't quit their day jobs, and most founders of successful ones do. If startup failure were a disease, the CDC would be issuing bulletins warning people to avoid day jobs.

Does that mean you should quit your day job? Not necessarily. I'm guessing here, but I'd guess that many of these would-be founders may not have the kind of determination it takes to start a company, and that in the back of their minds, they know it. The reason they don't invest more time in their startup is that they know it's a bad investment. [12]

I'd also guess there's some band of people who could have succeeded if they'd taken the leap and done it full-time, but didn't. I have no idea how wide this band is, but if the winner/borderline/hopeless progression has the sort of distribution you'd expect, the number of people who could have made it, if they'd quit their day job, is probably an order of magnitude larger than the number who do make it. [13]

If that's true, most startups that could succeed fail because the founders don't devote their whole efforts to them. That certainly accords with what I see out in the world. Most startups fail because they don't make something people want, and the reason most don't is that they don't try hard enough.

In other words, starting startups is just like everything else. The biggest mistake you can make is not to try hard enough. To the extent there's a secret to success, it's not to be in denial about that.

Notes

[1] This is not a complete list of the causes of failure, just those you can control. There are also several you can't, notably ineptitude and bad luck.

[2] Ironically, one variant of the Facebook that might work is a facebook exclusively for college students.

[3] Steve Jobs tried to motivate people by saying "Real artists ship." This is a fine sentence, but unfortunately not true. Many famous works of art are unfinished. It's true in fields that have hard deadlines, like architecture and filmmaking, but even there people tend to be tweaking stuff till it's yanked out of their hands.

[4] There's probably also a second factor: startup founders tend to be at the leading edge of technology, so problems they face are probably especially valuable.

[5] You should take more than you think you'll need, maybe 50% to 100% more, because software takes longer to write and deals longer to close than you expect.

[6] Since people sometimes call us VCs, I should add that we're not. VCs invest large amounts of other people's money. We invest small amounts of our own, like angel investors.

[7] Not linearly of course, or it would take forever to raise five million dollars. In practice it just feels like it takes forever.

Though if you include the cases where VCs don't invest, it would literally take forever in the median case. And maybe we should, because the danger of chasing large investments is not just that they take a long time. That's the best case. The real danger is that you'll expend a lot of time and get nothing.

[8] Some VCs will offer you an artificially low valuation to see if you have the balls to ask for more. It's lame that VCs play such games, but some do. If you're dealing with one of those you should push back on the valuation a bit.

[9] Suppose YouTube's founders had gone to Google in 2005 and told them "Google Video is badly designed. Give us $10 million and we'll tell you all the mistakes you made." They would have gotten the royal raspberry. Eighteen months later Google paid $1.6 billion for the same lesson, partly because they could then tell themselves that they were buying a phenomenon, or a community, or some vague thing like that.

I don't mean to be hard on Google. They did better than their competitors, who may have now missed the video boat entirely.

[10] Yes, actually: dealing with the government. But phone companies are up there.

[11] Many more than most people realize, because companies don't advertise this. Did you know Apple originally had three founders?

[12] I'm not dissing these people. I don't have the determination myself. I've twice come close to starting startups since Viaweb, and both times I bailed because I realized that without the spur of poverty I just wasn't willing to endure the stress of a startup.

[13] So how do you know whether you're in the category of people who should quit their day job, or the presumably larger one who shouldn't? I got to the point of saying that this was hard to judge for yourself and that you should seek outside advice, before realizing that that's what we do. We think of ourselves as investors, but viewed from the other direction Y Combinator is a service for advising people whether or not to quit their day job. We could be mistaken, and no doubt often are, but we do at least bet money on our conclusions.

The Hardest Lessons for Startups to Learn

April 2006

(This essay is derived from a talk at the 2006 Startup School.)

The startups we've funded so far are pretty quick, but they seem quicker to learn some lessons than others. I think it's because some things about startups are kind of counterintuitive.

We've now invested in enough companies that I've learned a trick for determining which points are the counterintuitive ones: they're the ones I have to keep repeating.

So I'm going to number these points, and maybe with future startups I'll be able to pull off a form of Huffman coding. I'll make them all read this, and then instead of nagging them in detail, I'll just be able to say: number four!

1. Release Early.

The thing I probably repeat most is this recipe for a startup: get a version 1 out fast, then improve it based on users' reactions.

By "release early" I don't mean you should release something full of bugs, but that you should release something minimal. Users hate bugs, but they don't seem to mind a minimal version 1, if there's more coming soon.

There are several reasons it pays to get version 1 done fast. One is that this is simply the right way to write software, whether for a startup or not. I've been repeating that since 1993, and I haven't seen much since to contradict it. I've seen a lot of startups die because they were too slow to release stuff, and none because they were too quick. [1]

One of the things that will surprise you if you build something popular is that you won't know your users. Reddit now has almost half a million unique visitors a month. Who are all those people? They have no idea. No web startup does. And since you don't know your users, it's dangerous to guess what they'll like. Better to release something and let them tell you.

Wufoo took this to heart and released their form-builder before the underlying database. You can't even drive the thing yet, but 83,000 people came to sit in the driver's seat and hold the steering wheel. And Wufoo got valuable

feedback from it: Linux users complained they used too much Flash, so they rewrote their software not to. If they'd waited to release everything at once, they wouldn't have discovered this problem till it was more deeply wired in.

Even if you had no users, it would still be important to release quickly, because for a startup the initial release acts as a shakedown cruise. If anything major is broken-- if the idea's no good, for example, or the founders hate one another-- the stress of getting that first version out will expose it. And if you have such problems you want to find them early.

Perhaps the most important reason to release early, though, is that it makes you work harder. When you're working on something that isn't released, problems are intriguing. In something that's out there, problems are alarming. There is a lot more urgency once you release. And I think that's precisely why people put it off. They know they'll have to work a lot harder once they do. [2]

2. Keep Pumping Out Features.

Of course, "release early" has a second component, without which it would be bad advice. If you're going to start with something that doesn't do much, you better improve it fast.

What I find myself repeating is "pump out features." And this rule isn't just for the initial stages. This is something all startups should do for as long as they want to be considered startups.

I don't mean, of course, that you should make your application ever more complex. By "feature" I mean one unit of hacking-- one quantum of making users' lives better.

As with exercise, improvements beget improvements. If you run every day, you'll probably feel like running tomorrow. But if you skip running for a couple weeks, it will be an effort to drag yourself out. So it is with hacking: the more ideas you implement, the more ideas you'll have. You should make your system better at least in some small way every day or two.

This is not just a good way to get development done; it is also a form of marketing. Users love a site that's constantly improving. In fact, users expect a site to improve. Imagine if you visited a site that seemed very good, and then returned two months later and not one thing had changed. Wouldn't it start to seem lame? [3]

They'll like you even better when you improve in response to their comments, because customers are used to companies ignoring them. If you're the rare exception-- a company that actually listens-- you'll generate fanatical loyalty. You won't need to advertise, because your users will do it for you.

This seems obvious too, so why do I have to keep repeating it? I think the problem here is that people get used to how things are. Once a product gets past the stage where it has glaring flaws, you start to get used to it, and

gradually whatever features it happens to have become its identity. For example, I doubt many people at Yahoo (or Google for that matter) realized how much better web mail could be till Paul Buchheit showed them.

I think the solution is to assume that anything you've made is far short of what it could be. Force yourself, as a sort of intellectual exercise, to keep thinking of improvements. Ok, sure, what you have is perfect. But if you had to change something, what would it be?

If your product seems finished, there are two possible explanations: (a) it is finished, or (b) you lack imagination. Experience suggests (b) is a thousand times more likely.

3. Make Users Happy.

Improving constantly is an instance of a more general rule: make users happy. One thing all startups have in common is that they can't force anyone to do anything. They can't force anyone to use their software, and they can't force anyone to do deals with them. A startup has to sing for its supper. That's why the successful ones make great things. They have to, or die.

When you're running a startup you feel like a little bit of debris blown about by powerful winds. The most powerful wind is users. They can either catch you and loft you up into the sky, as they did with Google, or leave you flat on the pavement, as they do with most startups. Users are a fickle wind, but more powerful than any other. If they take you up, no competitor can keep you down.

As a little piece of debris, the rational thing for you to do is not to lie flat, but to curl yourself into a shape the wind will catch.

I like the wind metaphor because it reminds you how impersonal the stream of traffic is. The vast majority of people who visit your site will be casual visitors. It's them you have to design your site for. The people who really care will find what they want by themselves.

The median visitor will arrive with their finger poised on the Back button. Think about your own experience: most links you follow lead to something lame. Anyone who has used the web for more than a couple weeks has been trained to click on Back after following a link. So your site has to say "Wait! Don't click on Back. This site isn't lame. Look at this, for example."

There are two things you have to do to make people pause. The most important is to explain, as concisely as possible, what the hell your site is about. How often have you visited a site that seemed to assume you already knew what they did? For example, the corporate site that says the company makes

enterprise content management solutions for business that enable organizations to unify people, content and processes to minimize business risk, accelerate time-to-value and sustain lower total cost of ownership.

An established company may get away with such an opaque description, but no startup can. A startup should be able to explain in one or two sentences exactly what it does. [4] And not just to users. You need this for everyone: investors, acquirers, partners, reporters, potential employees, and even current employees. You probably shouldn't even start a company to do something that can't be described compellingly in one or two sentences.

The other thing I repeat is to give people everything you've got, right away. If you have something impressive, try to put it on the front page, because that's the only one most visitors will see. Though indeed there's a paradox here: the more you push the good stuff toward the front, the more likely visitors are to explore further. [5]

In the best case these two suggestions get combined: you tell visitors what your site is about by showing them. One of the standard pieces of advice in fiction writing is "show, don't tell." Don't say that a character's angry; have him grind his teeth, or break his pencil in half. Nothing will explain what your site does so well as using it.

The industry term here is "conversion." The job of your site is to convert casual visitors into users-- whatever your definition of a user is. You can measure this in your growth rate. Either your site is catching on, or it isn't, and you must know which. If you have decent growth, you'll win in the end, no matter how obscure you are now. And if you don't, you need to fix something.

4. Fear the Right Things.

Another thing I find myself saying a lot is "don't worry." Actually, it's more often "don't worry about this; worry about that instead." Startups are right to be paranoid, but they sometimes fear the wrong things.

Most visible disasters are not so alarming as they seem. Disasters are normal in a startup: a founder quits, you discover a patent that covers what you're doing, your servers keep crashing, you run into an insoluble technical problem, you have to change your name, a deal falls through-- these are all par for the course. They won't kill you unless you let them.

Nor will most competitors. A lot of startups worry "what if Google builds something like us?" Actually big companies are not the ones you have to worry about-- not even Google. The people at Google are smart, but no smarter than you; they're not as motivated, because Google is not going to go out of business if this one product fails; and even at Google they have a lot of bureaucracy to slow them down.

What you should fear, as a startup, is not the established players, but other startups you don't know exist yet. They're way more dangerous than Google because, like you, they're cornered animals.

Looking just at existing competitors can give you a false sense of security. You should compete against what someone else could be doing, not just what you can see people doing. A corollary is that you shouldn't relax just because you have no visible competitors yet. No matter what your idea, there's someone else out there working on the same thing.

That's the downside of it being easier to start a startup: more people are doing it. But I disagree with Caterina Fake when she says that makes this a bad time to start a startup. More people are starting startups, but not as many more as could. Most college graduates still think they have to get a job. The average person can't ignore something that's been beaten into their head since they were three just because serving web pages recently got a lot cheaper.

And in any case, competitors are not the biggest threat. Way more startups hose themselves than get crushed by competitors. There are a lot of ways to do it, but the three main ones are internal disputes, inertia, and ignoring users. Each is, by itself, enough to kill you. But if I had to pick the worst, it would be ignoring users. If you want a recipe for a startup that's going to die, here it is: a couple of founders who have some great idea they know everyone is going to love, and that's what they're going to build, no matter what.

Almost everyone's initial plan is broken. If companies stuck to their initial plans, Microsoft would be selling programming languages, and Apple would be selling printed circuit boards. In both cases their customers told them what their business should be-- and they were smart enough to listen.

As Richard Feynman said, the imagination of nature is greater than the imagination of man. You'll find more interesting things by looking at the world than you could ever produce just by thinking. This principle is very powerful. It's why the best abstract painting still falls short of Leonardo, for example. And it applies to startups too. No idea for a product could ever be so clever as the ones you can discover by smashing a beam of prototypes into a beam of users.

5. Commitment Is a Self-Fulfilling Prophecy.

I now have enough experience with startups to be able to say what the most important quality is in a startup founder, and it's not what you might think. The most important quality in a startup founder is determination. Not intelligence-- determination.

This is a little depressing. I'd like to believe Viaweb succeeded because we were smart, not merely determined. A lot of people in the startup world want to believe that. Not just founders, but investors too. They like the idea of inhabiting a world ruled by intelligence. And you can tell they really believe this, because it affects their investment decisions.

Time after time VCs invest in startups founded by eminent professors. This may work in biotech, where a lot of startups simply commercialize existing research, but in software you want to invest in students, not professors. Microsoft, Yahoo, and Google were all founded by people who dropped out of school to do it. What students lack in experience they more than make up in dedication.

Of course, if you want to get rich, it's not enough merely to be determined. You have to be smart too, right? I'd like to think so, but I've had an experience that convinced me otherwise: I spent several years living in New York.

You can lose quite a lot in the brains department and it won't kill you. But lose even a little bit in the commitment department, and that will kill you very rapidly.

Running a startup is like walking on your hands: it's possible, but it requires extraordinary effort. If an ordinary employee were asked to do the things a startup founder has to, he'd be very indignant. Imagine if you were hired at some big company, and in addition to writing software ten times faster than you'd ever had to before, they expected you to answer support calls, administer the servers, design the web site, cold-call customers, find the company office space, and go out and get everyone lunch.

And to do all this not in the calm, womb-like atmosphere of a big company, but against a backdrop of constant disasters. That's the part that really demands determination. In a startup, there's always some disaster happening. So if you're the least bit inclined to find an excuse to quit, there's always one right there.

But if you lack commitment, chances are it will have been hurting you long before you actually quit. Everyone who deals with startups knows how important commitment is, so if they sense you're ambivalent, they won't give you much attention. If you lack commitment, you'll just find that for some mysterious reason good things happen to your competitors but not to you. If you lack commitment, it will seem to you that you're unlucky.

Whereas if you're determined to stick around, people will pay attention to you, because odds are they'll have to deal with you later. You're a local, not just a tourist, so everyone has to come to terms with you.

At Y Combinator we sometimes mistakenly fund teams who have the attitude that they're going to give this startup thing a shot for three months, and if something great happens, they'll stick with it-- "something great" meaning either that someone wants to buy them or invest millions of dollars in them. But if this is your attitude, "something great" is very unlikely to happen to you, because both acquirers and investors judge you by your level of commitment.

If an acquirer thinks you're going to stick around no matter what, they'll be more likely to buy you, because if they don't and you stick around, you'll probably grow, your price will go up, and they'll be left wishing they'd bought you earlier. Ditto for investors. What really motivates investors, even big VCs, is not the hope of good returns, but the fear of missing out. [6] So if you make it clear you're going to succeed no matter what, and the only reason you need them is to make it happen a little faster, you're much more likely to get money.

You can't fake this. The only way to convince everyone that you're ready to fight to the death is actually to be ready to.

You have to be the right kind of determined, though. I carefully chose the word determined rather than stubborn, because stubbornness is a disastrous quality in a startup. You have to be determined, but flexible, like a running back. A successful running back doesn't just put his head down and try to run through people. He improvises: if

someone appears in front of him, he runs around them; if someone tries to grab him, he spins out of their grip; he'll even run in the wrong direction briefly if that will help. The one thing he'll never do is stand still. [7]

6. There Is Always Room.

I was talking recently to a startup founder about whether it might be good to add a social component to their software. He said he didn't think so, because the whole social thing was tapped out. Really? So in a hundred years the only social networking sites will be the Facebook, MySpace, Flickr, and Del.icio.us? Not likely.

There is always room for new stuff. At every point in history, even the darkest bits of the dark ages, people were discovering things that made everyone say "why didn't anyone think of that before?" We know this continued to be true up till 2004, when the Facebook was founded-- though strictly speaking someone else did think of that.

The reason we don't see the opportunities all around us is that we adjust to however things are, and assume that's how things have to be. For example, it would seem crazy to most people to try to make a better search engine than Google. Surely that field, at least, is tapped out. Really? In a hundred years-- or even twenty-- are people still going to search for information using something like the current Google? Even Google probably doesn't think that.

In particular, I don't think there's any limit to the number of startups. Sometimes you hear people saying "All these guys starting startups now are going to be disappointed. How many little startups are Google and Yahoo going to buy, after all?" That sounds cleverly skeptical, but I can prove it's mistaken. No one proposes that there's some limit to the number of people who can be employed in an economy consisting of big, slow-moving companies with a couple thousand people each. Why should there be any limit to the number who could be employed by small, fast-moving companies with ten each? It seems to me the only limit would be the number of people who want to work that hard.

The limit on the number of startups is not the number that can get acquired by Google and Yahoo-- though it seems even that should be unlimited, if the startups were actually worth buying-- but the amount of wealth that can be created. And I don't think there's any limit on that, except cosmological ones.

So for all practical purposes, there is no limit to the number of startups. Startups make wealth, which means they make things people want, and if there's a limit on the number of things people want, we are nowhere near it. I still don't even have a flying car.

7. Don't Get Your Hopes Up.

This is another one I've been repeating since long before Y Combinator. It was practically the corporate motto at Viaweb.

Startup founders are naturally optimistic. They wouldn't do it otherwise. But you should treat your optimism the way you'd treat the core of a nuclear reactor: as a source of power that's also very dangerous. You have to build a shield around it, or it will fry you.

The shielding of a reactor is not uniform; the reactor would be useless if it were. It's pierced in a few places to let pipes in. An optimism shield has to be pierced too. I think the place to draw the line is between what you expect of yourself, and what you expect of other people. It's ok to be optimistic about what you can do, but assume the worst about machines and other people.

This is particularly necessary in a startup, because you tend to be pushing the limits of whatever you're doing. So things don't happen in the smooth, predictable way they do in the rest of the world. Things change suddenly, and usually for the worse.

Shielding your optimism is nowhere more important than with deals. If your startup is doing a deal, just assume it's not going to happen. The VCs who say they're going to invest in you aren't. The company that says they're going to buy you isn't. The big customer who wants to use your system in their whole company won't. Then if things work out you can be pleasantly surprised.

The reason I warn startups not to get their hopes up is not to save them from being disappointed when things fall through. It's for a more practical reason: to prevent them from leaning their company against something that's going to fall over, taking them with it.

For example, if someone says they want to invest in you, there's a natural tendency to stop looking for other investors. That's why people proposing deals seem so positive: they want you to stop looking. And you want to stop too, because doing deals is a pain. Raising money, in particular, is a huge time sink. So you have to consciously force yourself to keep looking.

Even if you ultimately do the first deal, it will be to your advantage to have kept looking, because you'll get better terms. Deals are dynamic; unless you're negotiating with someone unusually honest, there's not a single point where you shake hands and the deal's done. There are usually a lot of subsidiary questions to be cleared up after the handshake, and if the other side senses weakness-- if they sense you need this deal-- they will be very tempted to screw you in the details.

VCs and corp dev guys are professional negotiators. They're trained to take advantage of weakness. [8] So while they're often nice guys, they just can't help it. And as pros they do this more than you. So don't even try to bluff them. The only way a startup can have any leverage in a deal is genuinely not to need it. And if you don't believe in a deal, you'll be less likely to depend on it.

So I want to plant a hypnotic suggestion in your heads: when you hear someone say the words "we want to invest in you" or "we want to acquire you," I want the following phrase to appear automatically in your head: don't get your hopes up. Just continue running your company as if this deal didn't exist. Nothing is more likely to make it close.

The way to succeed in a startup is to focus on the goal of getting lots of users, and keep walking swiftly toward it while investors and acquirers scurry alongside trying to wave money in your face.

Speed, not Money

The way I've described it, starting a startup sounds pretty stressful. It is. When I talk to the founders of the companies we've funded, they all say the same thing: I knew it would be hard, but I didn't realize it would be this hard.

So why do it? It would be worth enduring a lot of pain and stress to do something grand or heroic, but just to make money? Is making money really that important?

No, not really. It seems ridiculous to me when people take business too seriously. I regard making money as a boring errand to be got out of the way as soon as possible. There is nothing grand or heroic about starting a startup per se.

So why do I spend so much time thinking about startups? I'll tell you why. Economically, a startup is best seen not as a way to get rich, but as a way to work faster. You have to make a living, and a startup is a way to get that done quickly, instead of letting it drag on through your whole life. [9]

We take it for granted most of the time, but human life is fairly miraculous. It is also palpably short. You're given this marvellous thing, and then poof, it's taken away. You can see why people invent gods to explain it. But even to people who don't believe in gods, life commands respect. There are times in most of our lives when the days go by in a blur, and almost everyone has a sense, when this happens, of wasting something precious. As Ben Franklin said, if you love life, don't waste time, because time is what life is made of.

So no, there's nothing particularly grand about making money. That's not what makes startups worth the trouble. What's important about startups is the speed. By compressing the dull but necessary task of making a living into the smallest possible time, you show respect for life, and there is something grand about that.

Notes

[1] Startups can die from releasing something full of bugs, and not fixing them fast enough, but I don't know of any that died from releasing something stable but minimal very early, then promptly improving it.

[2] I know this is why I haven't released Arc. The moment I do, I'll have people nagging me for features.

[3] A web site is different from a book or movie or desktop application in this respect. Users judge a site not as a single snapshot, but as an animation with multiple frames. Of the two, I'd say the rate of improvement is more important to users than where you currently are.

[4] It should not always tell this to users, however. For example, MySpace is basically a replacement mall for mallrats. But it was wiser for them, initially, to pretend that the site was about bands.

[5] Similarly, don't make users register to try your site. Maybe what you have is so valuable that visitors should gladly register to get at it. But they've been trained to expect the opposite. Most of the things they've tried on the web have sucked-- and probably especially those that made them register.

[6] VCs have rational reasons for behaving this way. They don't make their money (if they make money) off their median investments. In a typical fund, half the companies fail, most of the rest generate mediocre returns, and one or two "make the fund" by succeeding spectacularly. So if they miss just a few of the most promising opportunities, it could hose the whole fund.

[7] The attitude of a running back doesn't translate to soccer. Though it looks great when a forward dribbles past multiple defenders, a player who persists in trying such things will do worse in the long term than one who passes.

[8] The reason Y Combinator never negotiates valuations is that we're not professional negotiators, and don't want to turn into them.

[9] There are two ways to do work you love: (a) to make money, then work on what you love, or (b) to get a job where you get paid to work on stuff you love. In practice the first phases of both consist mostly of unedifying schleps, and in (b) the second phase is less secure.

The Hardest Lessons for Startups to Learn

April 2006

(This essay is derived from a talk at the 2006 Startup School.)

The startups we've funded so far are pretty quick, but they seem quicker to learn some lessons than others. I think it's because some things about startups are kind of counterintuitive.

We've now invested in enough companies that I've learned a trick for determining which points are the counterintuitive ones: they're the ones I have to keep repeating.

So I'm going to number these points, and maybe with future startups I'll be able to pull off a form of Huffman coding. I'll make them all read this, and then instead of nagging them in detail, I'll just be able to say: number four!

1. Release Early.

The thing I probably repeat most is this recipe for a startup: get a version 1 out fast, then improve it based on users' reactions.

By "release early" I don't mean you should release something full of bugs, but that you should release something minimal. Users hate bugs, but they don't seem to mind a minimal version 1, if there's more coming soon.

There are several reasons it pays to get version 1 done fast. One is that this is simply the right way to write software, whether for a startup or not. I've been repeating that since 1993, and I haven't seen much since to contradict it. I've seen a lot of startups die because they were too slow to release stuff, and none because they were too quick. [1]

One of the things that will surprise you if you build something popular is that you won't know your users. Reddit now has almost half a million unique visitors a month. Who are all those people? They have no idea. No web startup does. And since you don't know your users, it's dangerous to guess what they'll like. Better to release something and let them tell you.

Wufoo took this to heart and released their form-builder before the underlying database. You can't even drive the thing yet, but 83,000 people came to sit in the driver's seat and hold the steering wheel. And Wufoo got valuable

feedback from it: Linux users complained they used too much Flash, so they rewrote their software not to. If they'd waited to release everything at once, they wouldn't have discovered this problem till it was more deeply wired in.

Even if you had no users, it would still be important to release quickly, because for a startup the initial release acts as a shakedown cruise. If anything major is broken-- if the idea's no good, for example, or the founders hate one another-- the stress of getting that first version out will expose it. And if you have such problems you want to find them early.

Perhaps the most important reason to release early, though, is that it makes you work harder. When you're working on something that isn't released, problems are intriguing. In something that's out there, problems are alarming. There is a lot more urgency once you release. And I think that's precisely why people put it off. They know they'll have to work a lot harder once they do. [2]

2. Keep Pumping Out Features.

Of course, "release early" has a second component, without which it would be bad advice. If you're going to start with something that doesn't do much, you better improve it fast.

What I find myself repeating is "pump out features." And this rule isn't just for the initial stages. This is something all startups should do for as long as they want to be considered startups.

I don't mean, of course, that you should make your application ever more complex. By "feature" I mean one unit of hacking-- one quantum of making users' lives better.

As with exercise, improvements beget improvements. If you run every day, you'll probably feel like running tomorrow. But if you skip running for a couple weeks, it will be an effort to drag yourself out. So it is with hacking: the more ideas you implement, the more ideas you'll have. You should make your system better at least in some small way every day or two.

This is not just a good way to get development done; it is also a form of marketing. Users love a site that's constantly improving. In fact, users expect a site to improve. Imagine if you visited a site that seemed very good, and then returned two months later and not one thing had changed. Wouldn't it start to seem lame? [3]

They'll like you even better when you improve in response to their comments, because customers are used to companies ignoring them. If you're the rare exception-- a company that actually listens-- you'll generate fanatical loyalty. You won't need to advertise, because your users will do it for you.

This seems obvious too, so why do I have to keep repeating it? I think the problem here is that people get used to how things are. Once a product gets past the stage where it has glaring flaws, you start to get used to it, and

gradually whatever features it happens to have become its identity. For example, I doubt many people at Yahoo (or Google for that matter) realized how much better web mail could be till Paul Buchheit showed them.

I think the solution is to assume that anything you've made is far short of what it could be. Force yourself, as a sort of intellectual exercise, to keep thinking of improvements. Ok, sure, what you have is perfect. But if you had to change something, what would it be?

If your product seems finished, there are two possible explanations: (a) it is finished, or (b) you lack imagination. Experience suggests (b) is a thousand times more likely.

3. Make Users Happy.

Improving constantly is an instance of a more general rule: make users happy. One thing all startups have in common is that they can't force anyone to do anything. They can't force anyone to use their software, and they can't force anyone to do deals with them. A startup has to sing for its supper. That's why the successful ones make great things. They have to, or die.

When you're running a startup you feel like a little bit of debris blown about by powerful winds. The most powerful wind is users. They can either catch you and loft you up into the sky, as they did with Google, or leave you flat on the pavement, as they do with most startups. Users are a fickle wind, but more powerful than any other. If they take you up, no competitor can keep you down.

As a little piece of debris, the rational thing for you to do is not to lie flat, but to curl yourself into a shape the wind will catch.

I like the wind metaphor because it reminds you how impersonal the stream of traffic is. The vast majority of people who visit your site will be casual visitors. It's them you have to design your site for. The people who really care will find what they want by themselves.

The median visitor will arrive with their finger poised on the Back button. Think about your own experience: most links you follow lead to something lame. Anyone who has used the web for more than a couple weeks has been trained to click on Back after following a link. So your site has to say "Wait! Don't click on Back. This site isn't lame. Look at this, for example."

There are two things you have to do to make people pause. The most important is to explain, as concisely as possible, what the hell your site is about. How often have you visited a site that seemed to assume you already knew what they did? For example, the corporate site that says the company makes

enterprise content management solutions for business that enable organizations to unify people, content and processes to minimize business risk, accelerate time-to-value and sustain lower total cost of ownership.

An established company may get away with such an opaque description, but no startup can. A startup should be able to explain in one or two sentences exactly what it does. [4] And not just to users. You need this for everyone: investors, acquirers, partners, reporters, potential employees, and even current employees. You probably shouldn't even start a company to do something that can't be described compellingly in one or two sentences.

The other thing I repeat is to give people everything you've got, right away. If you have something impressive, try to put it on the front page, because that's the only one most visitors will see. Though indeed there's a paradox here: the more you push the good stuff toward the front, the more likely visitors are to explore further. [5]

In the best case these two suggestions get combined: you tell visitors what your site is about by showing them. One of the standard pieces of advice in fiction writing is "show, don't tell." Don't say that a character's angry; have him grind his teeth, or break his pencil in half. Nothing will explain what your site does so well as using it.

The industry term here is "conversion." The job of your site is to convert casual visitors into users-- whatever your definition of a user is. You can measure this in your growth rate. Either your site is catching on, or it isn't, and you must know which. If you have decent growth, you'll win in the end, no matter how obscure you are now. And if you don't, you need to fix something.

4. Fear the Right Things.

Another thing I find myself saying a lot is "don't worry." Actually, it's more often "don't worry about this; worry about that instead." Startups are right to be paranoid, but they sometimes fear the wrong things.

Most visible disasters are not so alarming as they seem. Disasters are normal in a startup: a founder quits, you discover a patent that covers what you're doing, your servers keep crashing, you run into an insoluble technical problem, you have to change your name, a deal falls through-- these are all par for the course. They won't kill you unless you let them.

Nor will most competitors. A lot of startups worry "what if Google builds something like us?" Actually big companies are not the ones you have to worry about-- not even Google. The people at Google are smart, but no smarter than you; they're not as motivated, because Google is not going to go out of business if this one product fails; and even at Google they have a lot of bureaucracy to slow them down.

What you should fear, as a startup, is not the established players, but other startups you don't know exist yet. They're way more dangerous than Google because, like you, they're cornered animals.

Looking just at existing competitors can give you a false sense of security. You should compete against what someone else could be doing, not just what you can see people doing. A corollary is that you shouldn't relax just because you have no visible competitors yet. No matter what your idea, there's someone else out there working on the same thing.

That's the downside of it being easier to start a startup: more people are doing it. But I disagree with Caterina Fake when she says that makes this a bad time to start a startup. More people are starting startups, but not as many more as could. Most college graduates still think they have to get a job. The average person can't ignore something that's been beaten into their head since they were three just because serving web pages recently got a lot cheaper.

And in any case, competitors are not the biggest threat. Way more startups hose themselves than get crushed by competitors. There are a lot of ways to do it, but the three main ones are internal disputes, inertia, and ignoring users. Each is, by itself, enough to kill you. But if I had to pick the worst, it would be ignoring users. If you want a recipe for a startup that's going to die, here it is: a couple of founders who have some great idea they know everyone is going to love, and that's what they're going to build, no matter what.

Almost everyone's initial plan is broken. If companies stuck to their initial plans, Microsoft would be selling programming languages, and Apple would be selling printed circuit boards. In both cases their customers told them what their business should be-- and they were smart enough to listen.

As Richard Feynman said, the imagination of nature is greater than the imagination of man. You'll find more interesting things by looking at the world than you could ever produce just by thinking. This principle is very powerful. It's why the best abstract painting still falls short of Leonardo, for example. And it applies to startups too. No idea for a product could ever be so clever as the ones you can discover by smashing a beam of prototypes into a beam of users.

5. Commitment Is a Self-Fulfilling Prophecy.

I now have enough experience with startups to be able to say what the most important quality is in a startup founder, and it's not what you might think. The most important quality in a startup founder is determination. Not intelligence-- determination.

This is a little depressing. I'd like to believe Viaweb succeeded because we were smart, not merely determined. A lot of people in the startup world want to believe that. Not just founders, but investors too. They like the idea of inhabiting a world ruled by intelligence. And you can tell they really believe this, because it affects their investment decisions.

Time after time VCs invest in startups founded by eminent professors. This may work in biotech, where a lot of startups simply commercialize existing research, but in software you want to invest in students, not professors. Microsoft, Yahoo, and Google were all founded by people who dropped out of school to do it. What students lack in experience they more than make up in dedication.

Of course, if you want to get rich, it's not enough merely to be determined. You have to be smart too, right? I'd like to think so, but I've had an experience that convinced me otherwise: I spent several years living in New York.

You can lose quite a lot in the brains department and it won't kill you. But lose even a little bit in the commitment department, and that will kill you very rapidly.

Running a startup is like walking on your hands: it's possible, but it requires extraordinary effort. If an ordinary employee were asked to do the things a startup founder has to, he'd be very indignant. Imagine if you were hired at some big company, and in addition to writing software ten times faster than you'd ever had to before, they expected you to answer support calls, administer the servers, design the web site, cold-call customers, find the company office space, and go out and get everyone lunch.

And to do all this not in the calm, womb-like atmosphere of a big company, but against a backdrop of constant disasters. That's the part that really demands determination. In a startup, there's always some disaster happening. So if you're the least bit inclined to find an excuse to quit, there's always one right there.

But if you lack commitment, chances are it will have been hurting you long before you actually quit. Everyone who deals with startups knows how important commitment is, so if they sense you're ambivalent, they won't give you much attention. If you lack commitment, you'll just find that for some mysterious reason good things happen to your competitors but not to you. If you lack commitment, it will seem to you that you're unlucky.

Whereas if you're determined to stick around, people will pay attention to you, because odds are they'll have to deal with you later. You're a local, not just a tourist, so everyone has to come to terms with you.

At Y Combinator we sometimes mistakenly fund teams who have the attitude that they're going to give this startup thing a shot for three months, and if something great happens, they'll stick with it-- "something great" meaning either that someone wants to buy them or invest millions of dollars in them. But if this is your attitude, "something great" is very unlikely to happen to you, because both acquirers and investors judge you by your level of commitment.

If an acquirer thinks you're going to stick around no matter what, they'll be more likely to buy you, because if they don't and you stick around, you'll probably grow, your price will go up, and they'll be left wishing they'd bought you earlier. Ditto for investors. What really motivates investors, even big VCs, is not the hope of good returns, but the fear of missing out. [6] So if you make it clear you're going to succeed no matter what, and the only reason you need them is to make it happen a little faster, you're much more likely to get money.

You can't fake this. The only way to convince everyone that you're ready to fight to the death is actually to be ready to.

You have to be the right kind of determined, though. I carefully chose the word determined rather than stubborn, because stubbornness is a disastrous quality in a startup. You have to be determined, but flexible, like a running back. A successful running back doesn't just put his head down and try to run through people. He improvises: if

someone appears in front of him, he runs around them; if someone tries to grab him, he spins out of their grip; he'll even run in the wrong direction briefly if that will help. The one thing he'll never do is stand still. [7]

6. There Is Always Room.

I was talking recently to a startup founder about whether it might be good to add a social component to their software. He said he didn't think so, because the whole social thing was tapped out. Really? So in a hundred years the only social networking sites will be the Facebook, MySpace, Flickr, and Del.icio.us? Not likely.

There is always room for new stuff. At every point in history, even the darkest bits of the dark ages, people were discovering things that made everyone say "why didn't anyone think of that before?" We know this continued to be true up till 2004, when the Facebook was founded-- though strictly speaking someone else did think of that.

The reason we don't see the opportunities all around us is that we adjust to however things are, and assume that's how things have to be. For example, it would seem crazy to most people to try to make a better search engine than Google. Surely that field, at least, is tapped out. Really? In a hundred years-- or even twenty-- are people still going to search for information using something like the current Google? Even Google probably doesn't think that.

In particular, I don't think there's any limit to the number of startups. Sometimes you hear people saying "All these guys starting startups now are going to be disappointed. How many little startups are Google and Yahoo going to buy, after all?" That sounds cleverly skeptical, but I can prove it's mistaken. No one proposes that there's some limit to the number of people who can be employed in an economy consisting of big, slow-moving companies with a couple thousand people each. Why should there be any limit to the number who could be employed by small, fast-moving companies with ten each? It seems to me the only limit would be the number of people who want to work that hard.

The limit on the number of startups is not the number that can get acquired by Google and Yahoo-- though it seems even that should be unlimited, if the startups were actually worth buying-- but the amount of wealth that can be created. And I don't think there's any limit on that, except cosmological ones.

So for all practical purposes, there is no limit to the number of startups. Startups make wealth, which means they make things people want, and if there's a limit on the number of things people want, we are nowhere near it. I still don't even have a flying car.

7. Don't Get Your Hopes Up.

This is another one I've been repeating since long before Y Combinator. It was practically the corporate motto at Viaweb.

Startup founders are naturally optimistic. They wouldn't do it otherwise. But you should treat your optimism the way you'd treat the core of a nuclear reactor: as a source of power that's also very dangerous. You have to build a shield around it, or it will fry you.

The shielding of a reactor is not uniform; the reactor would be useless if it were. It's pierced in a few places to let pipes in. An optimism shield has to be pierced too. I think the place to draw the line is between what you expect of yourself, and what you expect of other people. It's ok to be optimistic about what you can do, but assume the worst about machines and other people.

This is particularly necessary in a startup, because you tend to be pushing the limits of whatever you're doing. So things don't happen in the smooth, predictable way they do in the rest of the world. Things change suddenly, and usually for the worse.

Shielding your optimism is nowhere more important than with deals. If your startup is doing a deal, just assume it's not going to happen. The VCs who say they're going to invest in you aren't. The company that says they're going to buy you isn't. The big customer who wants to use your system in their whole company won't. Then if things work out you can be pleasantly surprised.

The reason I warn startups not to get their hopes up is not to save them from being disappointed when things fall through. It's for a more practical reason: to prevent them from leaning their company against something that's going to fall over, taking them with it.

For example, if someone says they want to invest in you, there's a natural tendency to stop looking for other investors. That's why people proposing deals seem so positive: they want you to stop looking. And you want to stop too, because doing deals is a pain. Raising money, in particular, is a huge time sink. So you have to consciously force yourself to keep looking.

Even if you ultimately do the first deal, it will be to your advantage to have kept looking, because you'll get better terms. Deals are dynamic; unless you're negotiating with someone unusually honest, there's not a single point where you shake hands and the deal's done. There are usually a lot of subsidiary questions to be cleared up after the handshake, and if the other side senses weakness-- if they sense you need this deal-- they will be very tempted to screw you in the details.

VCs and corp dev guys are professional negotiators. They're trained to take advantage of weakness. [8] So while they're often nice guys, they just can't help it. And as pros they do this more than you. So don't even try to bluff them. The only way a startup can have any leverage in a deal is genuinely not to need it. And if you don't believe in a deal, you'll be less likely to depend on it.

So I want to plant a hypnotic suggestion in your heads: when you hear someone say the words "we want to invest in you" or "we want to acquire you," I want the following phrase to appear automatically in your head: don't get your hopes up. Just continue running your company as if this deal didn't exist. Nothing is more likely to make it close.

The way to succeed in a startup is to focus on the goal of getting lots of users, and keep walking swiftly toward it while investors and acquirers scurry alongside trying to wave money in your face.

Speed, not Money

The way I've described it, starting a startup sounds pretty stressful. It is. When I talk to the founders of the companies we've funded, they all say the same thing: I knew it would be hard, but I didn't realize it would be this hard.

So why do it? It would be worth enduring a lot of pain and stress to do something grand or heroic, but just to make money? Is making money really that important?

No, not really. It seems ridiculous to me when people take business too seriously. I regard making money as a boring errand to be got out of the way as soon as possible. There is nothing grand or heroic about starting a startup per se.

So why do I spend so much time thinking about startups? I'll tell you why. Economically, a startup is best seen not as a way to get rich, but as a way to work faster. You have to make a living, and a startup is a way to get that done quickly, instead of letting it drag on through your whole life. [9]

We take it for granted most of the time, but human life is fairly miraculous. It is also palpably short. You're given this marvellous thing, and then poof, it's taken away. You can see why people invent gods to explain it. But even to people who don't believe in gods, life commands respect. There are times in most of our lives when the days go by in a blur, and almost everyone has a sense, when this happens, of wasting something precious. As Ben Franklin said, if you love life, don't waste time, because time is what life is made of.

So no, there's nothing particularly grand about making money. That's not what makes startups worth the trouble. What's important about startups is the speed. By compressing the dull but necessary task of making a living into the smallest possible time, you show respect for life, and there is something grand about that.

Notes

[1] Startups can die from releasing something full of bugs, and not fixing them fast enough, but I don't know of any that died from releasing something stable but minimal very early, then promptly improving it.

[2] I know this is why I haven't released Arc. The moment I do, I'll have people nagging me for features.

[3] A web site is different from a book or movie or desktop application in this respect. Users judge a site not as a single snapshot, but as an animation with multiple frames. Of the two, I'd say the rate of improvement is more important to users than where you currently are.

[4] It should not always tell this to users, however. For example, MySpace is basically a replacement mall for mallrats. But it was wiser for them, initially, to pretend that the site was about bands.

[5] Similarly, don't make users register to try your site. Maybe what you have is so valuable that visitors should gladly register to get at it. But they've been trained to expect the opposite. Most of the things they've tried on the web have sucked-- and probably especially those that made them register.

[6] VCs have rational reasons for behaving this way. They don't make their money (if they make money) off their median investments. In a typical fund, half the companies fail, most of the rest generate mediocre returns, and one or two "make the fund" by succeeding spectacularly. So if they miss just a few of the most promising opportunities, it could hose the whole fund.

[7] The attitude of a running back doesn't translate to soccer. Though it looks great when a forward dribbles past multiple defenders, a player who persists in trying such things will do worse in the long term than one who passes.

[8] The reason Y Combinator never negotiates valuations is that we're not professional negotiators, and don't want to turn into them.

[9] There are two ways to do work you love: (a) to make money, then work on what you love, or (b) to get a job where you get paid to work on stuff you love. In practice the first phases of both consist mostly of unedifying schleps, and in (b) the second phase is less secure.

How to Fund a Startup

Want to start a startup? Get funded by Y Combinator.

November 2005

Venture funding works like gears. A typical startup goes through several rounds of funding, and at each round you want to take just enough money to reach the speed where you can shift into the next gear.

Few startups get it quite right. Many are underfunded. A few are overfunded, which is like trying to start driving in third gear.

I think it would help founders to understand funding better—not just the mechanics of it, but what investors are thinking. I was surprised recently when I realized that all the worst problems we faced in our startup were due not to competitors, but investors. Dealing with competitors was easy by comparison.

I don't mean to suggest that our investors were nothing but a drag on us. They were helpful in negotiating deals, for example. I mean more that conflicts with investors are particularly nasty. Competitors punch you in the jaw, but investors have you by the balls.

Apparently our situation was not unusual. And if trouble with investors is one of the biggest threats to a startup, managing them is one of the most important skills founders need to learn.

Let's start by talking about the five sources of startup funding. Then we'll trace the life of a hypothetical (very fortunate) startup as it shifts gears through successive rounds.

Friends and Family

A lot of startups get their first funding from friends and family. Excite did, for example: after the founders graduated from college, they borrowed $15,000 from their parents to start a company. With the help of some part-time jobs they made it last 18 months.

If your friends or family happen to be rich, the line blurs between them and angel investors. At Viaweb we got our first $10,000 of seed money from our friend Julian, but he was sufficiently rich that it's hard to say whether he

should be classified as a friend or angel. He was also a lawyer, which was great, because it meant we didn't have to pay legal bills out of that initial small sum.

The advantage of raising money from friends and family is that they're easy to find. You already know them. There are three main disadvantages: you mix together your business and personal life; they will probably not be as well connected as angels or venture firms; and they may not be accredited investors, which could complicate your life later.

The SEC defines an "accredited investor" as someone with over a million dollars in liquid assets or an income of over $200,000 a year. The regulatory burden is much lower if a company's shareholders are all accredited investors. Once you take money from the general public you're more restricted in what you can do. [1]

A startup's life will be more complicated, legally, if any of the investors aren't accredited. In an IPO, it might not merely add expense, but change the outcome. A lawyer I asked about it said:

When the company goes public, the SEC will carefully study all prior issuances of stock by the company and demand that it take immediate action to cure any past violations of securities laws. Those remedial actions can delay, stall or even kill the IPO.

Of course the odds of any given startup doing an IPO are small. But not as small as they might seem. A lot of startups that end up going public didn't seem likely to at first. (Who could have guessed that the company Wozniak and Jobs started in their spare time selling plans for microcomputers would yield one of the biggest IPOs of the decade?) Much of the value of a startup consists of that tiny probability multiplied by the huge outcome.

It wasn't because they weren't accredited investors that I didn't ask my parents for seed money, though. When we were starting Viaweb, I didn't know about the concept of an accredited investor, and didn't stop to think about the value of investors' connections. The reason I didn't take money from my parents was that I didn't want them to lose it.

Consulting

Another way to fund a startup is to get a job. The best sort of job is a consulting project in which you can build whatever software you wanted to sell as a startup. Then you can gradually transform yourself from a consulting company into a product company, and have your clients pay your development expenses.

This is a good plan for someone with kids, because it takes most of the risk out of starting a startup. There never has to be a time when you have no revenues. Risk and reward are usually proportionate, however: you should expect a plan that cuts the risk of starting a startup also to cut the average return. In this case, you trade decreased financial risk for increased risk that your company won't succeed as a startup.

But isn't the consulting company itself a startup? No, not generally. A company has to be more than small and newly founded to be a startup. There are millions of small businesses in America, but only a few thousand are startups. To be a startup, a company has to be a product business, not a service business. By which I mean not that it has to make something physical, but that it has to have one thing it sells to many people, rather than doing custom work for individual clients. Custom work doesn't scale. To be a startup you need to be the band that sells a million copies of a song, not the band that makes money by playing at individual weddings and bar mitzvahs.

The trouble with consulting is that clients have an awkward habit of calling you on the phone. Most startups operate close to the margin of failure, and the distraction of having to deal with clients could be enough to put you over the edge. Especially if you have competitors who get to work full time on just being a startup.

So you have to be very disciplined if you take the consulting route. You have to work actively to prevent your company growing into a "weed tree," dependent on this source of easy but low-margin money. [2]

Indeed, the biggest danger of consulting may be that it gives you an excuse for failure. In a startup, as in grad school, a lot of what ends up driving you are the expectations of your family and friends. Once you start a startup and tell everyone that's what you're doing, you're now on a path labelled "get rich or bust." You now have to get rich, or you've failed.

Fear of failure is an extraordinarily powerful force. Usually it prevents people from starting things, but once you publish some definite ambition, it switches directions and starts working in your favor. I think it's a pretty clever piece of jiujitsu to set this irresistible force against the slightly less immovable object of becoming rich. You won't have it driving you if your stated ambition is merely to start a consulting company that you will one day morph into a startup.

An advantage of consulting, as a way to develop a product, is that you know you're making something at least one customer wants. But if you have what it takes to start a startup you should have sufficient vision not to need this crutch.

Angel Investors

Angels are individual rich people. The word was first used for backers of Broadway plays, but now applies to individual investors generally. Angels who've made money in technology are preferable, for two reasons: they understand your situation, and they're a source of contacts and advice.

The contacts and advice can be more important than the money. When del.icio.us took money from investors, they took money from, among others, Tim O'Reilly. The amount he put in was small compared to the VCs who led the round, but Tim is a smart and influential guy and it's good to have him on your side.

You can do whatever you want with money from consulting or friends and family. With angels we're now talking about venture funding proper, so it's time to introduce the concept of exit strategy. Younger would-be founders are often surprised that investors expect them either to sell the company or go public. The reason is that investors need to get their capital back. They'll only consider companies that have an exit strategy—meaning companies that could get bought or go public.

This is not as selfish as it sounds. There are few large, private technology companies. Those that don't fail all seem to get bought or go public. The reason is that employees are investors too—of their time—and they want just as much to be able to cash out. If your competitors offer employees stock options that might make them rich, while you make it clear you plan to stay private, your competitors will get the best people. So the principle of an "exit" is not just something forced on startups by investors, but part of what it means to be a startup.

Another concept we need to introduce now is valuation. When someone buys shares in a company, that implicitly establishes a value for it. If someone pays $20,000 for 10% of a company, the company is in theory worth $200,000. I say "in theory" because in early stage investing, valuations are voodoo. As a company gets more established, its valuation gets closer to an actual market value. But in a newly founded startup, the valuation number is just an artifact of the respective contributions of everyone involved.

Startups often "pay" investors who will help the company in some way by letting them invest at low valuations. If I had a startup and Steve Jobs wanted to invest in it, I'd give him the stock for $10, just to be able to brag that he was an investor. Unfortunately, it's impractical (if not illegal) to adjust the valuation of the company up and down for each investor. Startups' valuations are supposed to rise over time. So if you're going to sell cheap stock to eminent angels, do it early, when it's natural for the company to have a low valuation.

Some angel investors join together in syndicates. Any city where people start startups will have one or more of them. In Boston the biggest is the Common Angels. In the Bay Area it's the Band of Angels. You can find groups near you through the Angel Capital Association. [3] However, most angel investors don't belong to these groups. In fact, the more prominent the angel, the less likely they are to belong to a group.

Some angel groups charge you money to pitch your idea to them. Needless to say, you should never do this.

One of the dangers of taking investment from individual angels, rather than through an angel group or investment firm, is that they have less reputation to protect. A big-name VC firm will not screw you too outrageously, because other founders would avoid them if word got out. With individual angels you don't have this protection, as we found to our dismay in our own startup. In many startups' lives there comes a point when you're at the investors' mercy—when you're out of money and the only place to get more is your existing investors. When we got into such a scrape, our investors took advantage of it in a way that a name-brand VC probably wouldn't have.

Angels have a corresponding advantage, however: they're also not bound by all the rules that VC firms are. And so they can, for example, allow founders to cash out partially in a funding round, by selling some of their stock directly to the investors. I think this will become more common; the average founder is eager to do it, and selling, say, half a million dollars worth of stock will not, as VCs fear, cause most founders to be any less committed to the business.

The same angels who tried to screw us also let us do this, and so on balance I'm grateful rather than angry. (As in families, relations between founders and investors can be complicated.)

The best way to find angel investors is through personal introductions. You could try to cold-call angel groups near you, but angels, like VCs, will pay more attention to deals recommended by someone they respect.

Deal terms with angels vary a lot. There are no generally accepted standards. Sometimes angels' deal terms are as fearsome as VCs'. Other angels, particularly in the earliest stages, will invest based on a two-page agreement.

Angels who only invest occasionally may not themselves know what terms they want. They just want to invest in this startup. What kind of anti-dilution protection do they want? Hell if they know. In these situations, the deal terms tend to be random: the angel asks his lawyer to create a vanilla agreement, and the terms end up being whatever the lawyer considers vanilla. Which in practice usually means, whatever existing agreement he finds lying around his firm. (Few legal documents are created from scratch.)

These heaps o' boilerplate are a problem for small startups, because they tend to grow into the union of all preceding documents. I know of one startup that got from an angel investor what amounted to a five hundred pound handshake: after deciding to invest, the angel presented them with a 70-page agreement. The startup didn't have enough money to pay a lawyer even to read it, let alone negotiate the terms, so the deal fell through.

One solution to this problem would be to have the startup's lawyer produce the agreement, instead of the angel's. Some angels might balk at this, but others would probably welcome it.

Inexperienced angels often get cold feet when the time comes to write that big check. In our startup, one of the two angels in the initial round took months to pay us, and only did after repeated nagging from our lawyer, who was also, fortunately, his lawyer.

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 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. [4]

Seed Funding Firms

Seed firms are like angels in that they invest relatively small amounts at early stages, but like VCs in that they're companies that do it as a business, rather than individuals making occasional investments on the side.

Till now, nearly all seed firms have been so-called "incubators," so Y Combinator gets called one too, though the only thing we have in common is that we invest in the earliest phase.

According to the National Association of Business Incubators, there are about 800 incubators in the US. This is an astounding number, because I know the founders of a lot of startups, and I can't think of one that began in an incubator.

What is an incubator? I'm not sure myself. The defining quality seems to be that you work in their space. That's where the name "incubator" comes from. They seem to vary a great deal in other respects. At one extreme is the sort of pork-barrel project where a town gets money from the state government to renovate a vacant building as a "high-tech incubator," as if it were merely lack of the right sort of office space that had till now prevented the town from becoming a startup hub. At the other extreme are places like Idealab, which generates ideas for new startups internally and hires people to work for them.

The classic Bubble incubators, most of which now seem to be dead, were like VC firms except that they took a much bigger role in the startups they funded. In addition to working in their space, you were supposed to use their office staff, lawyers, accountants, and so on.

Whereas incubators tend (or tended) to exert more control than VCs, Y Combinator exerts less. And we think it's better if startups operate out of their own premises, however crappy, than the offices of their investors. So it's annoying that we keep getting called an "incubator," but perhaps inevitable, because there's only one of us so far and no word yet for what we are. If we have to be called something, the obvious name would be "excubator." (The name is more excusable if one considers it as meaning that we enable people to escape cubicles.)

Because seed firms are companies rather than individual people, reaching them is easier than reaching angels. Just go to their web site and send them an email. The importance of personal introductions varies, but is less than with angels or VCs.

The fact that seed firms are companies also means the investment process is more standardized. (This is generally true with angel groups too.) Seed firms will probably have set deal terms they use for every startup they fund. The

fact that the deal terms are standard doesn't mean they're favorable to you, but if other startups have signed the same agreements and things went well for them, it's a sign the terms are reasonable.

Seed firms differ from angels and VCs in that they invest exclusively in the earliest phases—often when the company is still just an idea. Angels and even VC firms occasionally do this, but they also invest at later stages.

The problems are different in the early stages. For example, in the first couple months a startup may completely redefine their idea. So seed investors usually care less about the idea than the people. This is true of all venture funding, but especially so in the seed stage.

Like VCs, one of the advantages of seed firms is the advice they offer. But because seed firms operate in an earlier phase, they need to offer different kinds of advice. For example, a seed firm should be able to give advice about how to approach VCs, which VCs obviously don't need to do; whereas VCs should be able to give advice about how to hire an "executive team," which is not an issue in the seed stage.

In the earliest phases, a lot of the problems are technical, so seed firms should be able to help with technical as well as business problems.

Seed firms and angel investors generally want to invest in the initial phases of a startup, then hand them off to VC firms for the next round. Occasionally startups go from seed funding direct to acquisition, however, and I expect this to become increasingly common.

Google has been aggressively pursuing this route, and now Yahoo is too. Both now compete directly with VCs. And this is a smart move. Why wait for further funding rounds to jack up a startup's price? When a startup reaches the point where VCs have enough information to invest in it, the acquirer should have enough information to buy it. More information, in fact; with their technical depth, the acquirers should be better at picking winners than VCs.

Venture Capital Funds

VC firms are like seed firms in that they're actual companies, but they invest other people's money, and much larger amounts of it. VC investments average several million dollars. So they tend to come later in the life of a startup, are harder to get, and come with tougher terms.

The word "venture capitalist" is sometimes used loosely for any venture investor, but there is a sharp difference between VCs and other investors: VC firms are organized as funds, much like hedge funds or mutual funds. The fund managers, who are called "general partners," get about 2% of the fund annually as a management fee, plus about 20% of the fund's gains.

There is a very sharp dropoff in performance among VC firms, because in the VC business both success and failure are self-perpetuating. When an investment scores spectacularly, as Google did for Kleiner and Sequoia, it generates a lot of good publicity for the VCs. And many founders prefer to take money from successful VC firms, because of the legitimacy it confers. Hence a vicious (for the losers) cycle: VC firms that have been doing badly will only get the deals the bigger fish have rejected, causing them to continue to do badly.

As a result, of the thousand or so VC funds in the US now, only about 50 are likely to make money, and it is very hard for a new fund to break into this group.

In a sense, the lower-tier VC firms are a bargain for founders. They may not be quite as smart or as well connected as the big-name firms, but they are much hungrier for deals. This means you should be able to get better terms from them.

Better how? The most obvious is valuation: they'll take less of your company. But as well as money, there's power. I think founders will increasingly be able to stay on as CEO, and on terms that will make it fairly hard to fire them later.

The most dramatic change, I predict, is that VCs will allow founders to cash out partially by selling some of their stock direct to the VC firm. VCs have traditionally resisted letting founders get anything before the ultimate "liquidity event." But they're also desperate for deals. And since I know from my own experience that the rule against buying stock from founders is a stupid one, this is a natural place for things to give as venture funding becomes more and more a seller's market.

The disadvantage of taking money from less known firms is that people will assume, correctly or not, that you were turned down by the more exalted ones. But, like where you went to college, the name of your VC stops mattering once you have some performance to measure. So the more confident you are, the less you need a brand-name VC. We funded Viaweb entirely with angel money; it never occurred to us that the backing of a well known VC firm would make us seem more impressive. [5]

Another danger of less known firms is that, like angels, they have less reputation to protect. I suspect it's the lower-tier firms that are responsible for most of the tricks that have given VCs such a bad reputation among hackers. They are doubly hosed: the general partners themselves are less able, and yet they have harder problems to solve, because the top VCs skim off all the best deals, leaving the lower-tier firms exactly the startups that are likely to blow up.

For example, lower-tier firms are much more likely to pretend to want to do a deal with you just to lock you up while they decide if they really want to. One experienced CFO said:

The better ones usually will not give a term sheet unless they really want to do a deal. The second or third tier firms have a much higher break rate—it could be as high as 50%.

It's obvious why: the lower-tier firms' biggest fear, when chance throws them a bone, is that one of the big dogs will notice and take it away. The big dogs don't have to worry about that.

Falling victim to this trick could really hurt you. As one VC told me:

If you were talking to four VCs, told three of them that you accepted a term sheet, and then have to call them back to tell them you were just kidding, you are absolutely damaged goods.

Here's a partial solution: when a VC offers you a term sheet, ask how many of their last 10 term sheets turned into deals. This will at least force them to lie outright if they want to mislead you.

Not all the people who work at VC firms are partners. Most firms also have a handful of junior employees called something like associates or analysts. If you get a call from a VC firm, go to their web site and check whether the person you talked to is a partner. Odds are it will be a junior person; they scour the web looking for startups their bosses could invest in. The junior people will tend to seem very positive about your company. They're not pretending; they want to believe you're a hot prospect, because it would be a huge coup for them if their firm invested in a company they discovered. Don't be misled by this optimism. It's the partners who decide, and they view things with a colder eye.

Because VCs invest large amounts, the money comes with more restrictions. Most only come into effect if the company gets into trouble. For example, VCs generally write it into the deal that in any sale, they get their investment back first. So if the company gets sold at a low price, the founders could get nothing. Some VCs now require that in any sale they get 4x their investment back before the common stock holders (that is, you) get anything, but this is an abuse that should be resisted.

Another difference with large investments is that the founders are usually required to accept "vesting"—to surrender their stock and earn it back over the next 4-5 years. VCs don't want to invest millions in a company the founders could just walk away from. Financially, vesting has little effect, but in some situations it could mean founders will have less power. If VCs got de facto control of the company and fired one of the founders, he'd lose any unvested stock unless there was specific protection against this. So vesting would in that situation force founders to toe the line.

The most noticeable change when a startup takes serious funding is that the founders will no longer have complete control. Ten years ago VCs used to insist that founders step down as CEO and hand the job over to a business guy they supplied. This is less the rule now, partly because the disasters of the Bubble showed that generic business guys don't make such great CEOs.

But while founders will increasingly be able to stay on as CEO, they'll have to cede some power, because the board of directors will become more powerful. In the seed stage, the board is generally a formality; if you want to talk to the other board members, you just yell into the next room. This stops with VC-scale money. In a typical VC funding deal, the board of directors might be composed of two VCs, two founders, and one outside person acceptable to both. The board will have ultimate power, which means the founders now have to convince instead of commanding.

This is not as bad as it sounds, however. Bill Gates is in the same position; he doesn't have majority control of Microsoft; in principle he also has to convince instead of commanding. And yet he seems pretty commanding,

doesn't he? As long as things are going smoothly, boards don't interfere much. The danger comes when there's a bump in the road, as happened to Steve Jobs at Apple.

Like angels, VCs prefer to invest in deals that come to them through people they know. So while nearly all VC funds have some address you can send your business plan to, VCs privately admit the chance of getting funding by this route is near zero. One recently told me that he did not know a single startup that got funded this way.

I suspect VCs accept business plans "over the transom" more as a way to keep tabs on industry trends than as a source of deals. In fact, I would strongly advise against mailing your business plan randomly to VCs, because they treat this as evidence of laziness. Do the extra work of getting personal introductions. As one VC put it:

I'm not hard to find. I know a lot of people. If you can't find some way to reach me, how are you going to create a successful company?

One of the most difficult problems for startup founders is deciding when to approach VCs. You really only get one chance, because they rely heavily on first impressions. And you can't approach some and save others for later, because (a) they ask who else you've talked to and when and (b) they talk among themselves. If you're talking to one VC and he finds out that you were rejected by another several months ago, you'll definitely seem shopworn.

So when do you approach VCs? When you can convince them. If the founders have impressive resumes and the idea isn't hard to understand, you could approach VCs quite early. Whereas if the founders are unknown and the idea is very novel, you might have to launch the thing and show that users loved it before VCs would be convinced.

If several VCs are interested in you, they will sometimes be willing to split the deal between them. They're more likely to do this if they're close in the VC pecking order. Such deals may be a net win for founders, because you get multiple VCs interested in your success, and you can ask each for advice about the other. One founder I know wrote:

Two-firm deals are great. It costs you a little more equity, but being able to play the two firms off each other (as well as ask one if the other is being out of line) is invaluable.

When you do negotiate with VCs, remember that they've done this a lot more than you have. They've invested in dozens of startups, whereas this is probably the first you've founded. But don't let them or the situation intimidate you. The average founder is smarter than the average VC. So just do what you'd do in any complex, unfamiliar situation: proceed deliberately, and question anything that seems odd.

It is, unfortunately, common for VCs to put terms in an agreement whose consequences surprise founders later, and also common for VCs to defend things they do by saying that they're standard in the industry. Standard, schmandard; the whole industry is only a few decades old, and rapidly evolving. The concept of "standard" is a useful one when you're operating on a small scale (Y Combinator uses identical terms for every deal because for tiny seed-stage investments it's not worth the overhead of negotiating individual deals), but it doesn't apply at the VC level. On that scale, every negotiation is unique.

Most successful startups get money from more than one of the preceding five sources. [6] And, confusingly, the names of funding sources also tend to be used as the names of different rounds. The best way to explain how it all works is to follow the case of a hypothetical startup.

Stage 1: Seed Round

Our startup begins when a group of three friends have an idea-- either an idea for something they might build, or simply the idea "let's start a company." Presumably they already have some source of food and shelter. But if you have food and shelter, you probably also have something you're supposed to be working on: either classwork, or a job. So if you want to work full-time on a startup, your money situation will probably change too.

A lot of startup founders say they started the company without any idea of what they planned to do. This is actually less common than it seems: many have to claim they thought of the idea after quitting because otherwise their former employer would own it.

The three friends decide to take the leap. Since most startups are in competitive businesses, you not only want to work full-time on them, but more than full-time. So some or all of the friends quit their jobs or leave school. (Some of the founders in a startup can stay in grad school, but at least one has to make the company his full-time job.)

They're going to run the company out of one of their apartments at first, and since they don't have any users they don't have to pay much for infrastructure. Their main expenses are setting up the company, which costs a couple thousand dollars in legal work and registration fees, and the living expenses of the founders.

The phrase "seed investment" covers a broad range. To some VC firms it means $500,000, but to most startups it means several months' living expenses. We'll suppose our group of friends start with $15,000 from their friend's rich uncle, who they give 5% of the company in return. There's only common stock at this stage. They leave 20% as an options pool for later employees (but they set things up so that they can issue this stock to themselves if they get bought early and most is still unissued), and the three founders each get 25%.

By living really cheaply they think they can make the remaining money last five months. When you have five months' runway left, how soon do you need to start looking for your next round? Answer: immediately. It takes time to find investors, and time (always more than you expect) for the deal to close even after they say yes. So if our group of founders know what they're doing they'll start sniffing around for angel investors right away. But of course their main job is to build version 1 of their software.

The friends might have liked to have more money in this first phase, but being slightly underfunded teaches them an important lesson. For a startup, cheapness is power. The lower your costs, the more options you have—not just at this stage, but at every point till you're profitable. When you have a high "burn rate," you're always under time pressure, which means (a) you don't have time for your ideas to evolve, and (b) you're often forced to take deals you don't like.

Every startup's rule should be: spend little, and work fast.

After ten weeks' work the three friends have built a prototype that gives one a taste of what their product will do. It's not what they originally set out to do—in the process of writing it, they had some new ideas. And it only does a fraction of what the finished product will do, but that fraction includes stuff that no one else has done before.

They've also written at least a skeleton business plan, addressing the five fundamental questions: what they're going to do, why users need it, how large the market is, how they'll make money, and who the competitors are and why this company is going to beat them. (That last has to be more specific than "they suck" or "we'll work really hard.")

If you have to choose between spending time on the demo or the business plan, spend most on the demo. Software is not only more convincing, but a better way to explore ideas.

Stage 2: Angel Round

While writing the prototype, the group has been traversing their network of friends in search of angel investors. They find some just as the prototype is demoable. When they demo it, one of the angels is willing to invest. Now the group is looking for more money: they want enough to last for a year, and maybe to hire a couple friends. So they're going to raise $200,000.

The angel agrees to invest at a pre-money valuation of $1 million. The company issues $200,000 worth of new shares to the angel; if there were 1000 shares before the deal, this means 200 additional shares. The angel now owns 200/1200 shares, or a sixth of the company, and all the previous shareholders' percentage ownership is diluted by a sixth. After the deal, the capitalization table looks like this:

shareholder	shares	percent
angel	200	16.7
uncle	50	4.2
each founder	250	20.8
option pool	200	16.7
total	1200	100

To keep things simple, I had the angel do a straight cash for stock deal. In reality the angel might be more likely to make the investment in the form of a convertible loan. A convertible loan is a loan that can be converted into stock later; it works out the same as a stock purchase in the end, but gives the angel more protection against being squashed by VCs in future rounds.

Who pays the legal bills for this deal? The startup, remember, only has a couple thousand left. In practice this turns out to be a sticky problem that usually gets solved in some improvised way. Maybe the startup can find lawyers who will do it cheaply in the hope of future work if the startup succeeds. Maybe someone has a lawyer friend. Maybe the angel pays for his lawyer to represent both sides. (Make sure if you take the latter route that the lawyer is representing you rather than merely advising you, or his only duty is to the investor.)

An angel investing $200k would probably expect a seat on the board of directors. He might also want preferred stock, meaning a special class of stock that has some additional rights over the common stock everyone else has. Typically these rights include vetoes over major strategic decisions, protection against being diluted in future rounds, and the right to get one's investment back first if the company is sold.

Some investors might expect the founders to accept vesting for a sum this size, and others wouldn't. VCs are more likely to require vesting than angels. At Viaweb we managed to raise $2.5 million from angels without ever accepting vesting, largely because we were so inexperienced that we were appalled at the idea. In practice this turned out to be good, because it made us harder to push around.

Our experience was unusual; vesting is the norm for amounts that size. Y Combinator doesn't require vesting, because (a) we invest such small amounts, and (b) we think it's unnecessary, and that the hope of getting rich is enough motivation to keep founders at work. But maybe if we were investing millions we would think differently.

I should add that vesting is also a way for founders to protect themselves against one another. It solves the problem of what to do if one of the founders quits. So some founders impose it on themselves when they start the company.

The angel deal takes two weeks to close, so we are now three months into the life of the company.

The point after you get the first big chunk of angel money will usually be the happiest phase in a startup's life. It's a lot like being a postdoc: you have no immediate financial worries, and few responsibilities. You get to work on juicy kinds of work, like designing software. You don't have to spend time on bureaucratic stuff, because you haven't hired any bureaucrats yet. Enjoy it while it lasts, and get as much done as you can, because you will never again be so productive.

With an apparently inexhaustible sum of money sitting safely in the bank, the founders happily set to work turning their prototype into something they can release. They hire one of their friends—at first just as a consultant, so they can try him out—and then a month later as employee #1. They pay him the smallest salary he can live on, plus 3% of the company in restricted stock, vesting over four years. (So after this the option pool is down to 13.7%). [7] They also spend a little money on a freelance graphic designer.

How much stock do you give early employees? That varies so much that there's no conventional number. If you get someone really good, really early, it might be wise to give him as much stock as the founders. The one universal rule is that the amount of stock an employee gets decreases polynomially with the age of the company. In other words,

you get rich as a power of how early you were. So if some friends want you to come work for their startup, don't wait several months before deciding.

A month later, at the end of month four, our group of founders have something they can launch. Gradually through word of mouth they start to get users. Seeing the system in use by real users—people they don't know—gives them lots of new ideas. Also they find they now worry obsessively about the status of their server. (How relaxing founders' lives must have been when startups wrote VisiCalc.)

By the end of month six, the system is starting to have a solid core of features, and a small but devoted following. People start to write about it, and the founders are starting to feel like experts in their field.

We'll assume that their startup is one that could put millions more to use. Perhaps they need to spend a lot on marketing, or build some kind of expensive infrastructure, or hire highly paid salesmen. So they decide to start talking to VCs. They get introductions to VCs from various sources: their angel investor connects them with a couple; they meet a few at conferences; a couple VCs call them after reading about them.

Step 3: Series A Round

Armed with their now somewhat fleshed-out business plan and able to demo a real, working system, the founders visit the VCs they have introductions to. They find the VCs intimidating and inscrutable. They all ask the same question: who else have you pitched to? (VCs are like high school girls: they're acutely aware of their position in the VC pecking order, and their interest in a company is a function of the interest other VCs show in it.)

One of the VC firms says they want to invest and offers the founders a term sheet. A term sheet is a summary of what the deal terms will be when and if they do a deal; lawyers will fill in the details later. By accepting the term sheet, the startup agrees to turn away other VCs for some set amount of time while this firm does the "due diligence" required for the deal. Due diligence is the corporate equivalent of a background check: the purpose is to uncover any hidden bombs that might sink the company later, like serious design flaws in the product, pending lawsuits against the company, intellectual property issues, and so on. VCs' legal and financial due diligence is pretty thorough, but the technical due diligence is generally a joke. [8]

The due diligence discloses no ticking bombs, and six weeks later they go ahead with the deal. Here are the terms: a $2 million investment at a pre-money valuation of $4 million, meaning that after the deal closes the VCs will own a third of the company (2 / (4 + 2)). The VCs also insist that prior to the deal the option pool be enlarged by an additional hundred shares. So the total number of new shares issued is 750, and the cap table becomes:

shareholder	shares	percent
VCs	650	33.3

angel 200 10.3

uncle 50 2.6

each founder 250 12.8

employee 36* 1.8 *unvested

option pool 264 13.5

---- -----

total 1950 100

This picture is unrealistic in several respects. For example, while the percentages might end up looking like this, it's unlikely that the VCs would keep the existing numbers of shares. In fact, every bit of the startup's paperwork would probably be replaced, as if the company were being founded anew. Also, the money might come in several tranches, the later ones subject to various conditions—though this is apparently more common in deals with lower-tier VCs (whose lot in life is to fund more dubious startups) than with the top firms.

And of course any VCs reading this are probably rolling on the floor laughing at how my hypothetical VCs let the angel keep his 10.3 of the company. I admit, this is the Bambi version; in simplifying the picture, I've also made everyone nicer. In the real world, VCs regard angels the way a jealous husband feels about his wife's previous boyfriends. To them the company didn't exist before they invested in it. [9]

I don't want to give the impression you have to do an angel round before going to VCs. In this example I stretched things out to show multiple sources of funding in action. Some startups could go directly from seed funding to a VC round; several of the companies we've funded have.

The founders are required to vest their shares over four years, and the board is now reconstituted to consist of two VCs, two founders, and a fifth person acceptable to both. The angel investor cheerfully surrenders his board seat.

At this point there is nothing new our startup can teach us about funding—or at least, nothing good. [10] The startup will almost certainly hire more people at this point; those millions must be put to work, after all. The company may do additional funding rounds, presumably at higher valuations. They may if they are extraordinarily fortunate do an IPO, which we should remember is also in principle a round of funding, regardless of its de facto purpose. But that, if not beyond the bounds of possibility, is beyond the scope of this article.

Deals Fall Through

Anyone who's been through a startup will find the preceding portrait to be missing something: disasters. If there's one thing all startups have in common, it's that something is always going wrong. And nowhere more than in matters of funding.

For example, our hypothetical startup never spent more than half of one round before securing the next. That's more ideal than typical. Many startups—even successful ones—come close to running out of money at some point. Terrible things happen to startups when they run out of money, because they're designed for growth, not adversity.

But the most unrealistic thing about the series of deals I've described is that they all closed. In the startup world, closing is not what deals do. What deals do is fall through. If you're starting a startup you would do well to remember that. Birds fly; fish swim; deals fall through.

Why? Partly the reason deals seem to fall through so often is that you lie to yourself. You want the deal to close, so you start to believe it will. But even correcting for this, startup deals fall through alarmingly often—far more often than, say, deals to buy real estate. The reason is that it's such a risky environment. People about to fund or acquire a startup are prone to wicked cases of buyer's remorse. They don't really grasp the risk they're taking till the deal's about to close. And then they panic. And not just inexperienced angel investors, but big companies too.

So if you're a startup founder wondering why some angel investor isn't returning your phone calls, you can at least take comfort in the thought that the same thing is happening to other deals a hundred times the size.

The example of a startup's history that I've presented is like a skeleton—accurate so far as it goes, but needing to be fleshed out to be a complete picture. To get a complete picture, just add in every possible disaster.

A frightening prospect? In a way. And yet also in a way encouraging. The very uncertainty of startups frightens away almost everyone. People overvalue stability—especially young people, who ironically need it least. And so in starting a startup, as in any really bold undertaking, merely deciding to do it gets you halfway there. On the day of the race, most of the other runners won't show up.

Notes

[1] The aim of such regulations is to protect widows and orphans from crooked investment schemes; people with a million dollars in liquid assets are assumed to be able to protect themselves. The unintended consequence is that the investments that generate the highest returns, like hedge funds, are available only to the rich.

[2] Consulting is where product companies go to die. IBM is the most famous example. So starting as a consulting company is like starting out in the grave and trying to work your way up into the world of the living.

[3] If "near you" doesn't mean the Bay Area, Boston, or Seattle, consider moving. It's not a coincidence you haven't heard of many startups from Philadelphia.

[4] Investors are often compared to sheep. And they are like sheep, but that's a rational response to their situation. Sheep act the way they do for a reason. If all the other sheep head for a certain field, it's probably good grazing. And when a wolf appears, is he going to eat a sheep in the middle of the flock, or one near the edge?

[5] This was partly confidence, and partly simple ignorance. We didn't know ourselves which VC firms were the impressive ones. We thought software was all that mattered. But that turned out to be the right direction to be naive in: it's much better to overestimate than underestimate the importance of making a good product.

[6] I've omitted one source: government grants. I don't think these are even worth thinking about for the average startup. Governments may mean well when they set up grant programs to encourage startups, but what they give with one hand they take away with the other: the process of applying is inevitably so arduous, and the restrictions on what you can do with the money so burdensome, that it would be easier to take a job to get the money.

You should be especially suspicious of grants whose purpose is some kind of social engineering-- e.g. to encourage more startups to be started in Mississippi. Free money to start a startup in a place where few succeed is hardly free.

Some government agencies run venture funding groups, which make investments rather than giving grants. For example, the CIA runs a venture fund called In-Q-Tel that is modelled on private sector funds and apparently generates good returns. They would probably be worth approaching—if you don't mind taking money from the CIA.

[7] Options have largely been replaced with restricted stock, which amounts to the same thing. Instead of earning the right to buy stock, the employee gets the stock up front, and earns the right not to have to give it back. The shares set aside for this purpose are still called the "option pool."

[8] First-rate technical people do not generally hire themselves out to do due diligence for VCs. So the most difficult part for startup founders is often responding politely to the inane questions of the "expert" they send to look you over.

[9] VCs regularly wipe out angels by issuing arbitrary amounts of new stock. They seem to have a standard piece of casuistry for this situation: that the angels are no longer working to help the company, and so don't deserve to keep their stock. This of course reflects a willful misunderstanding of what investment means; like any investor, the angel is being compensated for risks he took earlier. By a similar logic, one could argue that the VCs should be deprived of their shares when the company goes public.

[10] One new thing the company might encounter is a down round, or a funding round at valuation lower than the previous round. Down rounds are bad news; it is generally the common stock holders who take the hit. Some of the most fearsome provisions in VC deal terms have to do with down rounds—like "full ratchet anti-dilution," which is as frightening as it sounds.

Founders are tempted to ignore these clauses, because they think the company will either be a big success or a complete bust. VCs know otherwise: it's not uncommon for startups to have moments of adversity before they ultimately succeed. So it's worth negotiating anti-dilution provisions, even though you don't think you need to, and VCs will try to make you feel that you're being gratuitously troublesome.

The Hacker's Guide to Investors

April 2007

(This essay is derived from a keynote talk at the 2007 ASES Summit at Stanford.)

The world of investors is a foreign one to most hackers—partly because investors are so unlike hackers, and partly because they tend to operate in secret. I've been dealing with this world for many years, both as a founder and an investor, and I still don't fully understand it.

In this essay I'm going to list some of the more surprising things I've learned about investors. Some I only learned in the past year.

Teaching hackers how to deal with investors is probably the second most important thing we do at Y Combinator. The most important thing for a startup is to make something good. But everyone knows that's important. The dangerous thing about investors is that hackers don't know how little they know about this strange world.

1. The investors are what make a startup hub.

About a year ago I tried to figure out what you'd need to reproduce Silicon Valley. I decided the critical ingredients were rich people and nerds—investors and founders. People are all you need to make technology, and all the other people will move.

If I had to narrow that down, I'd say investors are the limiting factor. Not because they contribute more to the startup, but simply because they're least willing to move. They're rich. They're not going to move to Albuquerque just because there are some smart hackers there they could invest in. Whereas hackers will move to the Bay Area to find investors.

2. Angel investors are the most critical.

There are several types of investors. The two main categories are angels and VCs: VCs invest other people's money, and angels invest their own.

Though they're less well known, the angel investors are probably the more critical ingredient in creating a silicon valley. Most companies that VCs invest in would never have made it that far if angels hadn't invested first. VCs say between half and three quarters of companies that raise series A rounds have taken some outside investment already. [1]

Angels are willing to fund riskier projects than VCs. They also give valuable advice, because (unlike VCs) many have been startup founders themselves.

Google's story shows the key role angels play. A lot of people know Google raised money from Kleiner and Sequoia. What most don't realize is how late. That VC round was a series B round; the premoney valuation was $75 million. Google was already a successful company at that point. Really, Google was funded with angel money.

It may seem odd that the canonical Silicon Valley startup was funded by angels, but this is not so surprising. Risk is always proportionate to reward. So the most successful startup of all is likely to have seemed an extremely risky bet at first, and that is exactly the kind VCs won't touch.

Where do angel investors come from? From other startups. So startup hubs like Silicon Valley benefit from something like the marketplace effect, but shifted in time: startups are there because startups were there.

3. Angels don't like publicity.

If angels are so important, why do we hear more about VCs? Because VCs like publicity. They need to market themselves to the investors who are their "customers"—the endowments and pension funds and rich families whose money they invest—and also to founders who might come to them for funding.

Angels don't need to market themselves to investors because they invest their own money. Nor do they want to market themselves to founders: they don't want random people pestering them with business plans. Actually, neither do VCs. Both angels and VCs get deals almost exclusively through personal introductions. [2]

The reason VCs want a strong brand is not to draw in more business plans over the transom, but so they win deals when competing against other VCs. Whereas angels are rarely in direct competition, because (a) they do fewer deals, (b) they're happy to split them, and (c) they invest at a point where the stream is broader.

4. Most investors, especially VCs, are not like founders.

Some angels are, or were, hackers. But most VCs are a different type of people: they're dealmakers.

If you're a hacker, here's a thought experiment you can run to understand why there are basically no hacker VCs: How would you like a job where you never got to make anything, but instead spent all your time listening to other people pitch (mostly terrible) projects, deciding whether to fund them, and sitting on their boards if you did? That would not be fun for most hackers. Hackers like to make things. This would be like being an administrator.

Because most VCs are a different species of people from founders, it's hard to know what they're thinking. If you're a hacker, the last time you had to deal with these guys was in high school. Maybe in college you walked past their fraternity on your way to the lab. But don't underestimate them. They're as expert in their world as you are in yours. What they're good at is reading people, and making deals work to their advantage. Think twice before you try to beat them at that.

5. Most investors are momentum investors.

Because most investors are dealmakers rather than technology people, they generally don't understand what you're doing. I knew as a founder that most VCs didn't get technology. I also knew some made a lot of money. And yet it never occurred to me till recently to put those two ideas together and ask "How can VCs make money by investing in stuff they don't understand?"

The answer is that they're like momentum investors. You can (or could once) make a lot of money by noticing sudden changes in stock prices. When a stock jumps upward, you buy, and when it suddenly drops, you sell. In effect you're insider trading, without knowing what you know. You just know someone knows something, and that's making the stock move.

This is how most venture investors operate. They don't try to look at something and predict whether it will take off. They win by noticing that something is taking off a little sooner than everyone else. That generates almost as good returns as actually being able to pick winners. They may have to pay a little more than they would if they got in at the very beginning, but only a little.

Investors always say what they really care about is the team. Actually what they care most about is your traffic, then what other investors think, then the team. If you don't yet have any traffic, they fall back on number 2, what other investors think. And this, as you can imagine, produces wild oscillations in the "stock price" of a startup. One week everyone wants you, and they're begging not to be cut out of the deal. But all it takes is for one big investor to cool on you, and the next week no one will return your phone calls. We regularly have startups go from hot to cold or cold to hot in a matter of days, and literally nothing has changed.

There are two ways to deal with this phenomenon. If you're feeling really confident, you can try to ride it. You can start by asking a comparatively lowly VC for a small amount of money, and then after generating interest there, ask more prestigious VCs for larger amounts, stirring up a crescendo of buzz, and then "sell" at the top. This is extremely risky, and takes months even if you succeed. I wouldn't try it myself. My advice is to err on the side of safety: when someone offers you a decent deal, just take it and get on with building the company. Startups win or lose based on the quality of their product, not the quality of their funding deals.

6. Most investors are looking for big hits.

Venture investors like companies that could go public. That's where the big returns are. They know the odds of any individual startup going public are small, but they want to invest in those that at least have a chance of going public.

Currently the way VCs seem to operate is to invest in a bunch of companies, most of which fail, and one of which is Google. Those few big wins compensate for losses on their other investments. What this means is that most VCs will only invest in you if you're a potential Google. They don't care about companies that are a safe bet to be acquired for $20 million. There needs to be a chance, however small, of the company becoming really big.

Angels are different in this respect. They're happy to invest in a company where the most likely outcome is a $20 million acquisition if they can do it at a low enough valuation. But of course they like companies that could go public too. So having an ambitious long-term plan pleases everyone.

If you take VC money, you have to mean it, because the structure of VC deals prevents early acquisitions. If you take VC money, they won't let you sell early.

7. VCs want to invest large amounts.

The fact that they're running investment funds makes VCs want to invest large amounts. A typical VC fund is now hundreds of millions of dollars. If $400 million has to be invested by 10 partners, they have to invest $40 million each. VCs usually sit on the boards of companies they fund. If the average deal size was $1 million, each partner would have to sit on 40 boards, which would not be fun. So they prefer bigger deals, where they can put a lot of money to work at once.

VCs don't regard you as a bargain if you don't need a lot of money. That may even make you less attractive, because it means their investment creates less of a barrier to entry for competitors.

Angels are in a different position because they're investing their own money. They're happy to invest small amounts—sometimes as little as $20,000—as long as the potential returns look good enough. So if you're doing something inexpensive, go to angels.

8. Valuations are fiction.

VCs admit that valuations are an artifact. They decide how much money you need and how much of the company they want, and those two constraints yield a valuation.

Valuations increase as the size of the investment does. A company that an angel is willing to put $50,000 into at a valuation of a million can't take $6 million from VCs at that valuation. That would leave the founders less than a seventh of the company between them (since the option pool would also come out of that seventh). Most VCs wouldn't want that, which is why you never hear of deals where a VC invests $6 million at a premoney valuation of $1 million.

If valuations change depending on the amount invested, that shows how far they are from reflecting any kind of value of the company.

Since valuations are made up, founders shouldn't care too much about them. That's not the part to focus on. In fact, a high valuation can be a bad thing. If you take funding at a premoney valuation of $10 million, you won't be selling the company for 20. You'll have to sell for over 50 for the VCs to get even a 5x return, which is low to them. More likely they'll want you to hold out for 100. But needing to get a high price decreases the chance of getting bought at all; many companies can buy you for $10 million, but only a handful for 100. And since a startup is like a pass/fail course for the founders, what you want to optimize is your chance of a good outcome, not the percentage of the company you keep.

So why do founders chase high valuations? They're tricked by misplaced ambition. They feel they've achieved more if they get a higher valuation. They usually know other founders, and if they get a higher valuation they can say "mine is bigger than yours." But funding is not the real test. The real test is the final outcome for the founder, and getting too high a valuation may just make a good outcome less likely.

The one advantage of a high valuation is that you get less dilution. But there is another less sexy way to achieve that: just take less money.

9. Investors look for founders like the current stars.

Ten years ago investors were looking for the next Bill Gates. This was a mistake, because Microsoft was a very anomalous startup. They started almost as a contract programming operation, and the reason they became huge was that IBM happened to drop the PC standard in their lap.

Now all the VCs are looking for the next Larry and Sergey. This is a good trend, because Larry and Sergey are closer to the ideal startup founders.

Historically investors thought it was important for a founder to be an expert in business. So they were willing to fund teams of MBAs who planned to use the money to pay programmers to build their product for them. This is like funding Steve Ballmer in the hope that the programmer he'll hire is Bill Gates—kind of backward, as the events of the Bubble showed. Now most VCs know they should be funding technical guys. This is more pronounced among the very top funds; the lamer ones still want to fund MBAs.

If you're a hacker, it's good news that investors are looking for Larry and Sergey. The bad news is, the only investors who can do it right are the ones who knew them when they were a couple of CS grad students, not the confident media stars they are today. What investors still don't get is how clueless and tentative great founders can seem at the very beginning.

10. The contribution of investors tends to be underestimated.

Investors do more for startups than give them money. They're helpful in doing deals and arranging introductions, and some of the smarter ones, particularly angels, can give good advice about the product.

In fact, I'd say what separates the great investors from the mediocre ones is the quality of their advice. Most investors give advice, but the top ones give good advice.

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.

This trend is compounded by the obsession that the press has with founders. In a company founded by two people, 10% of the ideas might come from the first guy they hire. Arguably they've done a bad job of hiring otherwise. And yet this guy will be almost entirely overlooked by the press.

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. [3]

11. VCs are afraid of looking bad.

I've been very surprised to discover how timid most VCs are. They seem to be afraid of looking bad to their partners, and perhaps also to the limited partners—the people whose money they invest.

You can measure this fear in how much less risk VCs are willing to take. You can tell they won't make investments for their fund that they might be willing to make themselves as angels. Though it's not quite accurate to say that VCs are less willing to take risks. They're less willing to do things that might look bad. That's not the same thing.

For example, most VCs would be very reluctant to invest in a startup founded by a pair of 18 year old hackers, no matter how brilliant, because if the startup failed their partners could turn on them and say "What, you invested $x million of our money in a pair of 18 year olds?" Whereas if a VC invested in a startup founded by three former

banking executives in their 40s who planned to outsource their product development—which to my mind is actually a lot riskier than investing in a pair of really smart 18 year olds—he couldn't be faulted, if it failed, for making such an apparently prudent investment.

As a friend of mine said, "Most VCs can't do anything that would sound bad to the kind of doofuses who run pension funds." Angels can take greater risks because they don't have to answer to anyone.

12. Being turned down by investors doesn't mean much.

Some founders are quite dejected when they get turned down by investors. They shouldn't take it so much to heart. To start with, investors are often wrong. It's hard to think of a successful startup that wasn't turned down by investors at some point. Lots of VCs rejected Google. So obviously the reaction of investors is not a very meaningful test.

Investors will often reject you for what seem to be superficial reasons. I read of one VC who turned down a startup simply because they'd given away so many little bits of stock that the deal required too many signatures to close. [4] The reason investors can get away with this is that they see so many deals. It doesn't matter if they underestimate you because of some surface imperfection, because the next best deal will be almost as good. Imagine picking out apples at a grocery store. You grab one with a little bruise. Maybe it's just a surface bruise, but why even bother checking when there are so many other unbruised apples to choose from?

Investors would be the first to admit they're often wrong. So when you get rejected by investors, don't think "we suck," but instead ask "do we suck?" Rejection is a question, not an answer.

13. Investors are emotional.

I've been surprised to discover how emotional investors can be. You'd expect them to be cold and calculating, or at least businesslike, but often they're not. I'm not sure if it's their position of power that makes them this way, or the large sums of money involved, but investment negotiations can easily turn personal. If you offend investors, they'll leave in a huff.

A while ago an eminent VC firm offered a series A round to a startup we'd seed funded. Then they heard a rival VC firm was also interested. They were so afraid that they'd be rejected in favor of this other firm that they gave the startup what's known as an "exploding termsheet." They had, I think, 24 hours to say yes or no, or the deal was off. Exploding termsheets are a somewhat dubious device, but not uncommon. What surprised me was their reaction when I called to talk about it. I asked if they'd still be interested in the startup if the rival VC didn't end up making an offer, and they said no. What rational basis could they have had for saying that? If they thought the startup was worth investing in, what difference should it make what some other VC thought? Surely it was their duty to their limited partners simply to invest in the best opportunities they found; they should be delighted if the other VC said no, because it would mean they'd overlooked a good opportunity. But of course there was no rational basis for their decision. They just couldn't stand the idea of taking this rival firm's rejects.

In this case the exploding termsheet was not (or not only) a tactic to pressure the startup. It was more like the high school trick of breaking up with someone before they can break up with you. In an earlier essay I said that VCs were a lot like high school girls. A few VCs have joked about that characterization, but none have disputed it.

14. The negotiation never stops till the closing.

Most deals, for investment or acquisition, happen in two phases. There's an initial phase of negotiation about the big questions. If this succeeds you get a termsheet, so called because it outlines the key terms of a deal. A termsheet is not legally binding, but it is a definite step. It's supposed to mean that a deal is going to happen, once the lawyers work out all the details. In theory these details are minor ones; by definition all the important points are supposed to be covered in the termsheet.

Inexperience and wishful thinking combine to make founders feel that when they have a termsheet, they have a deal. They want there to be a deal; everyone acts like they have a deal; so there must be a deal. But there isn't and may not be for several months. A lot can change for a startup in several months. It's not uncommon for investors and acquirers to get buyer's remorse. So you have to keep pushing, keep selling, all the way to the close. Otherwise all the "minor" details left unspecified in the termsheet will be interpreted to your disadvantage. The other side may even break the deal; if they do that, they'll usually seize on some technicality or claim you misled them, rather than admitting they changed their minds.

It can be hard to keep the pressure on an investor or acquirer all the way to the closing, because the most effective pressure is competition from other investors or acquirers, and these tend to drop away when you get a termsheet. You should try to stay as close friends as you can with these rivals, but the most important thing is just to keep up the momentum in your startup. The investors or acquirers chose you because you seemed hot. Keep doing whatever made you seem hot. Keep releasing new features; keep getting new users; keep getting mentioned in the press and in blogs.

15. Investors like to co-invest.

I've been surprised how willing investors are to split deals. You might think that if they found a good deal they'd want it all to themselves, but they seem positively eager to syndicate. This is understandable with angels; they invest on a smaller scale and don't like to have too much money tied up in any one deal. But VCs also share deals a lot. Why?

Partly I think this is an artifact of the rule I quoted earlier: after traffic, VCs care most what other VCs think. A deal that has multiple VCs interested in it is more likely to close, so of deals that close, more will have multiple investors.

There is one rational reason to want multiple VCs in a deal: Any investor who co-invests with you is one less investor who could fund a competitor. Apparently Kleiner and Sequoia didn't like splitting the Google deal, but it did at least

have the advantage, from each one's point of view, that there probably wouldn't be a competitor funded by the other. Splitting deals thus has similar advantages to confusing paternity.

But I think the main reason VCs like splitting deals is the fear of looking bad. If another firm shares the deal, then in the event of failure it will seem to have been a prudent choice—a consensus decision, rather than just the whim of an individual partner.

16. Investors collude.

Investing is not covered by antitrust law. At least, it better not be, because investors regularly do things that would be illegal otherwise. I know personally of cases where one investor has talked another out of making a competitive offer, using the promise of sharing future deals.

In principle investors are all competing for the same deals, but the spirit of cooperation is stronger than the spirit of competition. The reason, again, is that there are so many deals. Though a professional investor may have a closer relationship with a founder he invests in than with other investors, his relationship with the founder is only going to last a couple years, whereas his relationship with other firms will last his whole career. There isn't so much at stake in his interactions with other investors, but there will be a lot of them. Professional investors are constantly trading little favors.

Another reason investors stick together is to preserve the power of investors as a whole. So you will not, as of this writing, be able to get investors into an auction for your series A round. They'd rather lose the deal than establish a precedent of VCs competitively bidding against one another. An efficient startup funding market may be coming in the distant future; things tend to move in that direction; but it's certainly not here now.

17. Large-scale investors care about their portfolio, not any individual company.

The reason startups work so well is that everyone with power also has equity. The only way any of them can succeed is if they all do. This makes everyone naturally pull in the same direction, subject to differences of opinion about tactics.

The problem is, larger scale investors don't have exactly the same motivation. Close, but not identical. They don't need any given startup to succeed, like founders do, just their portfolio as a whole to. So in borderline cases the rational thing for them to do is to sacrifice unpromising startups.

Large-scale investors tend to put startups in three categories: successes, failures, and the "living dead"—companies that are plugging along but don't seem likely in the immediate future to get bought or go public. To the founders, "living dead" sounds harsh. These companies may be far from failures by ordinary standards. But they might as well be from a venture investor's point of view, and they suck up just as much time and attention as the successes. So if such a company has two possible strategies, a conservative one that's slightly more likely to work in the end, or a

risky one that within a short time will either yield a giant success or kill the company, VCs will push for the kill-or-cure option. To them the company is already a write-off. Better to have resolution, one way or the other, as soon as possible.

If a startup gets into real trouble, instead of trying to save it VCs may just sell it at a low price to another of their portfolio companies. Philip Greenspun said in Founders at Work that Ars Digita's VCs did this to them.

18. Investors have different risk profiles from founders.

Most people would rather a 100% chance of $1 million than a 20% chance of $10 million. Investors are rich enough to be rational and prefer the latter. So they'll always tend to encourage founders to keep rolling the dice. If a company is doing well, investors will want founders to turn down most acquisition offers. And indeed, most startups that turn down acquisition offers ultimately do better. But it's still hair-raising for the founders, because they might end up with nothing. When someone's offering to buy you for a price at which your stock is worth $5 million, saying no is equivalent to having $5 million and betting it all on one spin of the roulette wheel.

Investors will tell you the company is worth more. And they may be right. But that doesn't mean it's wrong to sell. Any financial advisor who put all his client's assets in the stock of a single, private company would probably lose his license for it.

More and more, investors are letting founders cash out partially. That should correct the problem. Most founders have such low standards that they'll feel rich with a sum that doesn't seem huge to investors. But this custom is spreading too slowly, because VCs are afraid of seeming irresponsible. No one wants to be the first VC to give someone fuck-you money and then actually get told "fuck you." But until this does start to happen, we know VCs are being too conservative.

19. Investors vary greatly.

Back when I was a founder I used to think all VCs were the same. And in fact they do all look the same. They're all what hackers call "suits." But since I've been dealing with VCs more I've learned that some suits are smarter than others.

They're also in a business where winners tend to keep winning and losers to keep losing. When a VC firm has been successful in the past, everyone wants funding from them, so they get the pick of all the new deals. The self-reinforcing nature of the venture funding market means that the top ten firms live in a completely different world from, say, the hundredth. As well as being smarter, they tend to be calmer and more upstanding; they don't need to do iffy things to get an edge, and don't want to because they have more brand to protect.

There are only two kinds of VCs you want to take money from, if you have the luxury of choosing: the "top tier" VCs, meaning about the top 20 or so firms, plus a few new ones that are not among the top 20 only because they haven't been around long enough.

It's particularly important to raise money from a top firm if you're a hacker, because they're more confident. That means they're less likely to stick you with a business guy as CEO, like VCs used to do in the 90s. If you seem smart and want to do it, they'll let you run the company.

20. Investors don't realize how much it costs to raise money from them.

Raising money is a huge time suck at just the point where startups can least afford it. It's not unusual for it to take five or six months to close a funding round. Six weeks is fast. And raising money is not just something you can leave running as a background process. When you're raising money, it's inevitably the main focus of the company. Which means building the product isn't.

Suppose a Y Combinator company starts talking to VCs after demo day, and is successful in raising money from them, closing the deal after a comparatively short 8 weeks. Since demo day occurs after 10 weeks, the company is now 18 weeks old. Raising money, rather than working on the product, has been the company's main focus for 44% of its existence. And mind you, this an example where things turned out well.

When a startup does return to working on the product after a funding round finally closes, it's as if they were returning to work after a months-long illness. They've lost most of their momentum.

Investors have no idea how much they damage the companies they invest in by taking so long to do it. But companies do. So there is a big opportunity here for a new kind of venture fund that invests smaller amounts at lower valuations, but promises to either close or say no very quickly. If there were such a firm, I'd recommend it to startups in preference to any other, no matter how prestigious. Startups live on speed and momentum.

21. Investors don't like to say no.

The reason funding deals take so long to close is mainly that investors can't make up their minds. VCs are not big companies; they can do a deal in 24 hours if they need to. But they usually let the initial meetings stretch out over a couple weeks. The reason is the selection algorithm I mentioned earlier. Most don't try to predict whether a startup will win, but to notice quickly that it already is winning. They care what the market thinks of you and what other VCs think of you, and they can't judge those just from meeting you.

Because they're investing in things that (a) change fast and (b) they don't understand, a lot of investors will reject you in a way that can later be claimed not to have been a rejection. Unless you know this world, you may not even realize you've been rejected. Here's a VC saying no:

We're really excited about your project, and we want to keep in close touch as you develop it further.

Translated into more straightforward language, this means: We're not investing in you, but we may change our minds if it looks like you're taking off. Sometimes they're more candid and say explicitly that they need to "see some traction." They'll invest in you if you start to get lots of users. But so would any VC. So all they're saying is that you're still at square 1.

Here's a test for deciding whether a VC's response was yes or no. Look down at your hands. Are you holding a termsheet?

22. You need investors.

Some founders say "Who needs investors?" Empirically the answer seems to be: everyone who wants to succeed. Practically every successful startup takes outside investment at some point.

Why? What the people who think they don't need investors forget is that they will have competitors. The question is not whether you need outside investment, but whether it could help you at all. If the answer is yes, and you don't take investment, then competitors who do will have an advantage over you. And in the startup world a little advantage can expand into a lot.

Mike Moritz famously said that he invested in Yahoo because he thought they had a few weeks' lead over their competitors. That may not have mattered quite so much as he thought, because Google came along three years later and kicked Yahoo's ass. But there is something in what he said. Sometimes a small lead can grow into the yes half of a binary choice.

Maybe as it gets cheaper to start a startup, it will start to be possible to succeed in a competitive market without outside funding. There are certainly costs to raising money. But as of this writing the empirical evidence says it's a net win.

23. Investors like it when you don't need them.

A lot of founders approach investors as if they needed their permission to start a company—as if it were like getting into college. But you don't need investors to start most companies; they just make it easier.

And in fact, investors greatly prefer it if you don't need them. What excites them, both consciously and unconsciously, is the sort of startup that approaches them saying "the train's leaving the station; are you in or out?" not the one saying "please can we have some money to start a company?"

Most investors are "bottoms" in the sense that the startups they like most are those that are rough with them. When Google stuck Kleiner and Sequoia with a $75 million premoney valuation, their reaction was probably "Ouch! That feels so good." And they were right, weren't they? That deal probably made them more than any other they've done.

The thing is, VCs are pretty good at reading people. So don't try to act tough with them unless you really are the next Google, or they'll see through you in a second. Instead of acting tough, what most startups should do is simply always have a backup plan. Always have some alternative plan for getting started if any given investor says no. Having one is the best insurance against needing one.

So you shouldn't start a startup that's expensive to start, because then you'll be at the mercy of investors. If you ultimately want to do something that will cost a lot, start by doing a cheaper subset of it, and expand your ambitions when and if you raise more money.

Apparently the most likely animals to be left alive after a nuclear war are cockroaches, because they're so hard to kill. That's what you want to be as a startup, initially. Instead of a beautiful but fragile flower that needs to have its stem in a plastic tube to support itself, better to be small, ugly, and indestructible.

Notes

[1] I may be underestimating VCs. They may play some behind the scenes role in IPOs, which you ultimately need if you want to create a silicon valley.

[2] A few VCs have an email address you can send your business plan to, but the number of startups that get funded this way is basically zero. You should always get a personal introduction—and to a partner, not an associate.

[3] Several people have told us that the most valuable thing about startup school was that they got to see famous startup founders and realized they were just ordinary guys. Though we're happy to provide this service, this is not generally the way we pitch startup school to potential speakers.

[4] Actually this sounds to me like a VC who got buyer's remorse, then used a technicality to get out of the deal. But it's telling that it even seemed a plausible excuse.

How to Present to Investors

Want to start a startup? Get funded by Y Combinator.

August 2006, rev. April 2007, September 2010

In a few days it will be Demo Day, when the startups we funded this summer present to investors. Y Combinator funds startups twice a year, in January and June. Ten weeks later we invite all the investors we know to hear them present what they've built so far.

Ten weeks is not much time. The average startup probably doesn't have much to show for itself after ten weeks. But the average startup fails. When you look at the ones that went on to do great things, you find a lot that began with someone pounding out a prototype in a week or two of nonstop work. Startups are a counterexample to the rule that haste makes waste.

(Too much money seems to be as bad for startups as too much time, so we don't give them much money either.)

A week before Demo Day, we have a dress rehearsal called Rehearsal Day. At other Y Combinator events we allow outside guests, but not at Rehearsal Day. No one except the other founders gets to see the rehearsals.

The presentations on Rehearsal Day are often pretty rough. But this is to be expected. We try to pick founders who are good at building things, not ones who are slick presenters. Some of the founders are just out of college, or even still in it, and have never spoken to a group of people they didn't already know.

So we concentrate on the basics. On Demo Day each startup will only get ten minutes, so we encourage them to focus on just two goals: (a) explain what you're doing, and (b) explain why users will want it.

That might sound easy, but it's not when the speakers have no experience presenting, and they're explaining technical matters to an audience that's mostly non-technical.

This situation is constantly repeated when startups present to investors: people who are bad at explaining, talking to people who are bad at understanding. Practically every successful startup, including stars like Google, presented at

some point to investors who didn't get it and turned them down. Was it because the founders were bad at presenting, or because the investors were obtuse? It's probably always some of both.

At the most recent Rehearsal Day, we four Y Combinator partners found ourselves saying a lot of the same things we said at the last two. So at dinner afterward we collected all our tips about presenting to investors. Most startups face similar challenges, so we hope these will be useful to a wider audience.

1. Explain what you're doing.

Investors' main question when judging a very early startup is whether you've made a compelling product. Before they can judge whether you've built a good x, they have to understand what kind of x you've built. They will get very frustrated if instead of telling them what you do, you make them sit through some kind of preamble.

Say what you're doing as soon as possible, preferably in the first sentence. "We're Jeff and Bob and we've built an easy to use web-based database. Now we'll show it to you and explain why people need this."

If you're a great public speaker you may be able to violate this rule. Last year one founder spent the whole first half of his talk on a fascinating analysis of the limits of the conventional desktop metaphor. He got away with it, but unless you're a captivating speaker, which most hackers aren't, it's better to play it safe.

2. Get rapidly to demo.

This section is now obsolete for YC founders presenting at Demo Day, because Demo Day presentations are now so short that they rarely include much if any demo. They seem to work just as well without, however, which makes me think I was wrong to emphasize demos so much before.

A demo explains what you've made more effectively than any verbal description. The only thing worth talking about first is the problem you're trying to solve and why it's important. But don't spend more than a tenth of your time on that. Then demo.

When you demo, don't run through a catalog of features. Instead start with the problem you're solving, and then show how your product solves it. Show features in an order driven by some kind of purpose, rather than the order in which they happen to appear on the screen.

If you're demoing something web-based, assume that the network connection will mysteriously die 30 seconds into your presentation, and come prepared with a copy of the server software running on your laptop.

3. Better a narrow description than a vague one.

One reason founders resist describing their projects concisely is that, at this early stage, there are all kinds of possibilities. The most concise descriptions seem misleadingly narrow. So for example a group that has built an easy web-based database might resist calling their applicaton that, because it could be so much more. In fact, it could be anything...

The problem is, as you approach (in the calculus sense) a description of something that could be anything, the content of your description approaches zero. If you describe your web-based database as "a system to allow people to collaboratively leverage the value of information," it will go in one investor ear and out the other. They'll just discard that sentence as meaningless boilerplate, and hope, with increasing impatience, that in the next sentence you'll actually explain what you've made.

Your primary goal is not to describe everything your system might one day become, but simply to convince investors you're worth talking to further. So approach this like an algorithm that gets the right answer by successive approximations. Begin with a description that's gripping but perhaps overly narrow, then flesh it out to the extent you can. It's the same principle as incremental development: start with a simple prototype, then add features, but at every point have working code. In this case, "working code" means a working description in the investor's head.

4. Don't talk and drive.

Have one person talk while another uses the computer. If the same person does both, they'll inevitably mumble downwards at the computer screen instead of talking clearly at the audience.

As long as you're standing near the audience and looking at them, politeness (and habit) compel them to pay attention to you. Once you stop looking at them to fuss with something on your computer, their minds drift off to the errands they have to run later.

5. Don't talk about secondary matters at length.

If you only have a few minutes, spend them explaining what your product does and why it's great. Second order issues like competitors or resumes should be single slides you go through quickly at the end. If you have impressive resumes, just flash them on the screen for 15 seconds and say a few words. For competitors, list the top 3 and explain in one sentence each what they lack that you have. And put this kind of thing at the end, after you've made it clear what you've built.

6. Don't get too deeply into business models.

It's good to talk about how you plan to make money, but mainly because it shows you care about that and have thought about it. Don't go into detail about your business model, because (a) that's not what smart investors care about in a brief presentation, and (b) any business model you have at this point is probably wrong anyway.

Recently a VC who came to speak at Y Combinator talked about a company he just invested in. He said their business model was wrong and would probably change three times before they got it right. The founders were experienced guys who'd done startups before and who'd just succeeded in getting millions from one of the top VC firms, and even their business model was crap. (And yet he invested anyway, because he expected it to be crap at this stage.)

If you're solving an important problem, you're going to sound a lot smarter talking about that than the business model. The business model is just a bunch of guesses, and guesses about stuff that's probably not your area of expertise. So don't spend your precious few minutes talking about crap when you could be talking about solid, interesting things you know a lot about: the problem you're solving and what you've built so far.

As well as being a bad use of time, if your business model seems spectacularly wrong, that will push the stuff you want investors to remember out of their heads. They'll just remember you as the company with the boneheaded plan for making money, rather than the company that solved that important problem.

7. Talk slowly and clearly at the audience.

Everyone at Rehearsal Day could see the difference between the people who'd been out in the world for a while and had presented to groups, and those who hadn't.

You need to use a completely different voice and manner talking to a roomful of people than you would in conversation. Everyday life gives you no practice in this. If you can't already do it, the best solution is to treat it as a consciously artificial trick, like juggling.

However, that doesn't mean you should talk like some kind of announcer. Audiences tune that out. What you need to do is talk in this artificial way, and yet make it seem conversational. (Writing is the same. Good writing is an elaborate effort to seem spontaneous.)

If you want to write out your whole presentation beforehand and memorize it, that's ok. That has worked for some groups in the past. But make sure to write something that sounds like spontaneous, informal speech, and deliver it that way too.

Err on the side of speaking slowly. At Rehearsal Day, one of the founders mentioned a rule actors use: if you feel you're speaking too slowly, you're speaking at about the right speed.

8. Have one person talk.

Startups often want to show that all the founders are equal partners. This is a good instinct; investors dislike unbalanced teams. But trying to show it by partitioning the presentation is going too far. It's distracting. You can demonstrate your respect for one another in more subtle ways. For example, when one of the groups presented at Demo Day, the more extroverted of the two founders did most of the talking, but he described his co-founder as the best hacker he'd ever met, and you could tell he meant it.

Pick the one or at most two best speakers, and have them do most of the talking.

Exception: If one of the founders is an expert in some specific technical field, it can be good for them to talk about that for a minute or so. This kind of "expert witness" can add credibility, even if the audience doesn't understand all the details. If Jobs and Wozniak had 10 minutes to present the Apple II, it might be a good plan to have Jobs speak for 9 minutes and have Woz speak for a minute in the middle about some of the technical feats he'd pulled off in the design. (Though of course if it were actually those two, Jobs would speak for the entire 10 minutes.)

9. Seem confident.

Between the brief time available and their lack of technical background, many in the audience will have a hard time evaluating what you're doing. Probably the single biggest piece of evidence, initially, will be your own confidence in it. You have to show you're impressed with what you've made.

And I mean show, not tell. Never say "we're passionate" or "our product is great." People just ignore that—or worse, write you off as bullshitters. Such messages must be implicit.

What you must not do is seem nervous and apologetic. If you've truly made something good, you're doing investors a favor by telling them about it. If you don't genuinely believe that, perhaps you ought to change what your company is doing. If you don't believe your startup has such promise that you'd be doing them a favor by letting them invest, why are you investing your time in it?

10. Don't try to seem more than you are.

Don't worry if your company is just a few months old and doesn't have an office yet, or your founders are technical people with no business experience. Google was like that once, and they turned out ok. Smart investors can see past such superficial flaws. They're not looking for finished, smooth presentations. They're looking for raw talent. All you need to convince them of is that you're smart and that you're onto something good. If you try too hard to conceal your rawness—by trying to seem corporate, or pretending to know about stuff you don't—you may just conceal your talent.

You can afford to be candid about what you haven't figured out yet. Don't go out of your way to bring it up (e.g. by having a slide about what might go wrong), but don't try to pretend either that you're further along than you are. If you're a hacker and you're presenting to experienced investors, they're probably better at detecting bullshit than you are at producing it.

11. Don't put too many words on slides.

When there are a lot of words on a slide, people just skip reading it. So look at your slides and ask of each word "could I cross this out?" This includes gratuitous clip art. Try to get your slides under 20 words if you can.

Don't read your slides. They should be something in the background as you face the audience and talk to them, not something you face and read to an audience sitting behind you.

Cluttered sites don't do well in demos, especially when they're projected onto a screen. At the very least, crank up the font size big enough to make all the text legible. But cluttered sites are bad anyway, so perhaps you should use this opportunity to make your design simpler.

12. Specific numbers are good.

If you have any kind of data, however preliminary, tell the audience. Numbers stick in people's heads. If you can claim that the median visitor generates 12 page views, that's great.

But don't give them more than four or five numbers, and only give them numbers specific to you. You don't need to tell them the size of the market you're in. Who cares, really, if it's 500 million or 5 billion a year? Talking about that is like an actor at the beginning of his career telling his parents how much Tom Hanks makes. Yeah, sure, but first you have to become Tom Hanks. The important part is not whether he makes ten million a year or a hundred, but how you get there.

13. Tell stories about users.

The biggest fear of investors looking at early stage startups is that you've built something based on your own a priori theories of what the world needs, but that no one will actually want. So it's good if you can talk about problems specific users have and how you solve them.

Greg Mcadoo said one thing Sequoia looks for is the "proxy for demand." What are people doing now, using inadequate tools, that shows they need what you're making?

Another sign of user need is when people pay a lot for something. It's easy to convince investors there will be demand for a cheaper alternative to something popular, if you preserve the qualities that made it popular.

The best stories about user needs are about your own. A remarkable number of famous startups grew out of some need the founders had: Apple, Microsoft, Yahoo, Google. Experienced investors know that, so stories of this type will get their attention. The next best thing is to talk about the needs of people you know personally, like your friends or siblings.

14. Make a soundbite stick in their heads.

Professional investors hear a lot of pitches. After a while they all blur together. The first cut is simply to be one of those they remember. And the way to ensure that is to create a descriptive phrase about yourself that sticks in their heads.

In Hollywood, these phrases seem to be of the form "x meets y." In the startup world, they're usually "the x of y" or "the x y." Viaweb's was "the Microsoft Word of ecommerce."

Find one and launch it clearly (but apparently casually) in your talk, preferably near the beginning.

It's a good exercise for you, too, to sit down and try to figure out how to describe your startup in one compelling phrase. If you can't, your plans may not be sufficiently focused.

The Equity Equation

July 2007

An investor wants to give you money for a certain percentage of your startup. Should you take it? You're about to hire your first employee. How much stock should you give him?

These are some of the hardest questions founders face. And yet both have the same answer:

$1/(1 - n)$

Whenever you're trading stock in your company for anything, whether it's money or an employee or a deal with another company, the test for whether to do it is the same. You should give up n% of your company if what you trade it for improves your average outcome enough that the (100 - n)% you have left is worth more than the whole company was before.

For example, if an investor wants to buy half your company, how much does that investment have to improve your average outcome for you to break even? Obviously it has to double: if you trade half your company for something that more than doubles the company's average outcome, you're net ahead. You have half as big a share of something worth more than twice as much.

In the general case, if n is the fraction of the company you're giving up, the deal is a good one if it makes the company worth more than $1/(1 - n)$.

For example, suppose Y Combinator offers to fund you in return for 6% of your company. In this case, n is .06 and $1/(1 - n)$ is 1.064. So you should take the deal if you believe we can improve your average outcome by more than 6.4%. If we improve your outcome by 10%, you're net ahead, because the remaining .94 you hold is worth .94 x 1.1 = 1.034. [1]

One of the things the equity equation shows us is that, financially at least, taking money from a top VC firm can be a really good deal. Greg Mcadoo from Sequoia recently said at a YC dinner that when Sequoia invests alone they like to take about 30% of a company. 1/.7 = 1.43, meaning that deal is worth taking if they can improve your outcome by more than 43%. For the average startup, that would be an extraordinary bargain. It would improve the average startup's prospects by more than 43% just to be able to say they were funded by Sequoia, even if they never actually got the money.

The reason Sequoia is such a good deal is that the percentage of the company they take is artificially low. They don't even try to get market price for their investment; they limit their holdings to leave the founders enough stock to feel the company is still theirs.

The catch is that Sequoia gets about 6000 business plans a year and funds about 20 of them, so the odds of getting this great deal are 1 in 300. The companies that make it through are not average startups.

Of course, there are other factors to consider in a VC deal. It's never just a straight trade of money for stock. But if it were, taking money from a top firm would generally be a bargain.

You can use the same formula when giving stock to employees, but it works in the other direction. If i is the average outcome for the company with the addition of some new person, then they're worth n such that i = 1/(1 - n). Which means n = (i - 1)/i.

For example, suppose you're just two founders and you want to hire an additional hacker who's so good you feel he'll increase the average outcome of the whole company by 20%. n = (1.2 - 1)/1.2 = .167. So you'll break even if you trade 16.7% of the company for him.

That doesn't mean 16.7% is the right amount of stock to give him. Stock is not the only cost of hiring someone: there's usually salary and overhead as well. And if the company merely breaks even on the deal, there's no reason to do it.

I think to translate salary and overhead into stock you should multiply the annual rate by about 1.5. Most startups grow fast or die; if you die you don't have to pay the guy, and if you grow fast you'll be paying next year's salary out of next year's valuation, which should be 3x this year's. If your valuation grows 3x a year, the total cost in stock of a new hire's salary and overhead is 1.5 years' cost at the present valuation. [2]

How much of an additional margin should the company need as the "activation energy" for the deal? Since this is in effect the company's profit on a hire, the market will determine that: if you're a hot opportunity, you can charge more.

Let's run through an example. Suppose the company wants to make a "profit" of 50% on the new hire mentioned above. So subtract a third from 16.7% and we have 11.1% as his "retail" price. Suppose further that he's going to cost $60k a year in salary and overhead, x 1.5 = $90k total. If the company's valuation is $2 million, $90k is 4.5%. 11.1% - 4.5% = an offer of 6.6%.

Incidentally, notice how important it is for early employees to take little salary. It comes right out of stock that could otherwise be given to them.

Obviously there is a great deal of play in these numbers. I'm not claiming that stock grants can now be reduced to a formula. Ultimately you always have to guess. But at least know what you're guessing. If you choose a number based on your gut feel, or a table of typical grant sizes supplied by a VC firm, understand what those are estimates of.

And more generally, when you make any decision involving equity, run it through 1/(1 - n) to see if it makes sense. You should always feel richer after trading equity. If the trade didn't increase the value of your remaining shares enough to put you net ahead, you wouldn't have (or shouldn't have) done it.

Notes

[1] This is why we can't believe anyone would think Y Combinator was a bad deal. Does anyone really think we're so useless that in three months we can't improve a startup's prospects by 6.4%?

[2] The obvious choice for your present valuation is the post-money valuation of your last funding round. This probably undervalues the company, though, because (a) unless your last round just happened, the company is presumably worth more, and (b) the valuation of an early funding round usually reflects some other contribution by the investors.

A Fundraising Survival Guide

Want to start a startup? Get funded by Y Combinator.

August 2008

Raising money is the second hardest part of starting a startup. The hardest part is making something people want: most startups that die, die because they didn't do that. But the second biggest cause of death is probably the difficulty of raising money. Fundraising is brutal.

One reason it's so brutal is simply the brutality of markets. People who've spent most of their lives in schools or big companies may not have been exposed to that. Professors and bosses usually feel some sense of responsibility toward you; if you make a valiant effort and fail, they'll cut you a break. Markets are less forgiving. Customers don't care how hard you worked, only whether you solved their problems.

Investors evaluate startups the way customers evaluate products, not the way bosses evaluate employees. If you're making a valiant effort and failing, maybe they'll invest in your next startup, but not this one.

But raising money from investors is harder than selling to customers, because there are so few of them. There's nothing like an efficient market. You're unlikely to have more than 10 who are interested; it's difficult to talk to more. So the randomness of any one investor's behavior can really affect you.

Problem number 3: investors are very random. All investors, including us, are by ordinary standards incompetent. We constantly have to make decisions about things we don't understand, and more often than not we're wrong.

And yet a lot is at stake. The amounts invested by different types of investors vary from five thousand dollars to fifty million, but the amount usually seems large for whatever type of investor it is. Investment decisions are big decisions.

That combination—making big decisions about things they don't understand—tends to make investors very skittish. VCs are notorious for leading founders on. Some of the more unscrupulous do it deliberately. But even the most well-intentioned investors can behave in a way that would seem crazy in everyday life. One day they're full of enthusiasm and seem ready to write you a check on the spot; the next they won't return your phone calls. They're not playing games with you. They just can't make up their minds. [1]

If that weren't bad enough, these wildly fluctuating nodes are all linked together. Startup investors all know one another, and (though they hate to admit it) the biggest factor in their opinion of you is the opinion of other investors. [2] Talk about a recipe for an unstable system. You get the opposite of the damping that the fear/greed balance usually produces in markets. No one is interested in a startup that's a "bargain" because everyone else hates it.

So the inefficient market you get because there are so few players is exacerbated by the fact that they act less than independently. The result is a system like some kind of primitive, multi-celled sea creature, where you irritate one extremity and the whole thing contracts violently.

Y Combinator is working to fix this. We're trying to increase the number of investors just as we're increasing the number of startups. We hope that as the number of both increases we'll get something more like an efficient market. As t approaches infinity, Demo Day approaches an auction.

Unfortunately, t is still very far from infinity. What does a startup do now, in the imperfect world we currently inhabit? The most important thing is not to let fundraising get you down. Startups live or die on morale. If you let the difficulty of raising money destroy your morale, it will become a self-fulfilling prophecy.

Bootstrapping (= Consulting)

Some would-be founders may by now be thinking, why deal with investors at all? If raising money is so painful, why do it?

One answer to that is obvious: because you need money to live on. It's a fine idea in principle to finance your startup with its own revenues, but you can't create instant customers. Whatever you make, you have to sell a certain amount to break even. It will take time to grow your sales to that point, and it's hard to predict, till you try, how long it will take.

We could not have bootstrapped Viaweb, for example. We charged quite a lot for our software—about $140 per user per month—but it was at least a year before our revenues would have covered even our paltry costs. We didn't have enough saved to live on for a year.

If you factor out the "bootstrapped" companies that were actually funded by their founders through savings or a day job, the remainder either (a) got really lucky, which is hard to do on demand, or (b) began life as consulting companies and gradually transformed themselves into product companies.

Consulting is the only option you can count on. But consulting is far from free money. It's not as painful as raising money from investors, perhaps, but the pain is spread over a longer period. Years, probably. And for many types of startup, that delay could be fatal. If you're working on something so unusual that no one else is likely to think of it, you can take your time. Joshua Schachter gradually built Delicious on the side while working on Wall Street. He got away with it because no one else realized it was a good idea. But if you were building something as obviously

necessary as online store software at about the same time as Viaweb, and you were working on it on the side while spending most of your time on client work, you were not in a good position.

Bootstrapping sounds great in principle, but this apparently verdant territory is one from which few startups emerge alive. The mere fact that bootstrapped startups tend to be famous on that account should set off alarm bells. If it worked so well, it would be the norm. [3]

Bootstrapping may get easier, because starting a company is getting cheaper. But I don't think we'll ever reach the point where most startups can do without outside funding. Technology tends to get dramatically cheaper, but living expenses don't.

The upshot is, you can choose your pain: either the short, sharp pain of raising money, or the chronic ache of consulting. For a given total amount of pain, raising money is the better choice, because new technology is usually more valuable now than later.

But although for most startups raising money will be the lesser evil, it's still a pretty big evil—so big that it can easily kill you. Not merely in the obvious sense that if you fail to raise money you might have to shut the company down, but because the process of raising money itself can kill you.

To survive it you need a set of techniques mostly orthogonal to the ones used in convincing investors, just as mountain climbers need to know survival techniques that are mostly orthogonal to those used in physically getting up and down mountains.

1. Have low expectations.

The reason raising money destroys so many startups' morale is not simply that it's hard, but that it's so much harder than they expected. What kills you is the disappointment. And the lower your expectations, the harder it is to be disappointed.

Startup founders tend to be optimistic. This can work well in technology, at least some of the time, but it's the wrong way to approach raising money. Better to assume investors will always let you down. Acquirers too, while we're at it. At YC one of our secondary mantras is "Deals fall through." No matter what deal you have going on, assume it will fall through. The predictive power of this simple rule is amazing.

There will be a tendency, as a deal progresses, to start to believe it will happen, and then to depend on it happening. You must resist this. Tie yourself to the mast. This is what kills you. Deals do not have a trajectory like most other human interactions, where shared plans solidify linearly over time. Deals often fall through at the last moment. Often the other party doesn't really think about what they want till the last moment. So you can't use your everyday intuitions about shared plans as a guide. When it comes to deals, you have to consciously turn them off and become pathologically cynical.

This is harder to do than it sounds. It's very flattering when eminent investors seem interested in funding you. It's easy to start to believe that raising money will be quick and straightforward. But it hardly ever is.

2. Keep working on your startup.

It sounds obvious to say that you should keep working on your startup while raising money. Actually this is hard to do. Most startups don't manage to.

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.

The best way to survive the distraction of meeting with investors is probably to partition the company: to pick one founder to deal with investors while the others keep the company going. This works better when a startup has 3 founders than 2, and better when the leader of the company is not also the lead developer. In the best case, the company keeps moving forward at about half speed.

That's the best case, though. More often than not the company comes to a standstill while raising money. And that is dangerous for so many reasons. Raising money always takes longer than you expect. What seems like it's going to be a 2 week interruption turns into a 4 month interruption. That can be very demoralizing. And worse still, it can make you less attractive to investors. They want to invest in companies that are dynamic. A company that hasn't done anything new in 4 months doesn't seem dynamic, so they start to lose interest. Investors rarely grasp this, but much of what they're responding to when they lose interest in a startup is the damage done by their own indecision.

The solution: put the startup first. Fit meetings with investors into the spare moments in your development schedule, rather than doing development in the spare moments between meetings with investors. If you keep the company moving forward—releasing new features, increasing traffic, doing deals, getting written about—those investor meetings are more likely to be productive. Not just because your startup will seem more alive, but also because it will be better for your own morale, which is one of the main ways investors judge you.

3. Be conservative.

As conditions get worse, the optimal strategy becomes more conservative. When things go well you can take risks; when things are bad you want to play it safe.

I advise approaching fundraising as if it were always going badly. The reason is that between your ability to delude yourself and the wildly unstable nature of the system you're dealing with, things probably either already are or could easily become much worse than they seem.

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?

Corollary: if an investor seems interested, don't just let them sit. You can't assume someone interested in investing will stay interested. In fact, you can't even tell (they can't even tell) if they're really interested till you try to convert that interest into money. So if you have hot prospect, either close them now or write them off. And unless you already have enough funding, that reduces to: close them now.

Startups don't win by getting great funding rounds, but by making great products. So finish raising money and get back to work.

4. Be flexible.

There are two questions VCs ask that you shouldn't answer: "Who else are you talking to?" and "How much are you trying to raise?"

VCs don't expect you to answer the first question. They ask it just in case. [4] They do seem to expect an answer to the second. But I don't think you should just tell them a number. Not as a way to play games with them, but because you shouldn't have a fixed amount you need to raise.

The custom of a startup needing a fixed amount of funding is an obsolete one left over from the days when startups were more expensive. A company that needed to build a factory or hire 50 people obviously needed to raise a certain minimum amount. But few technology startups are in that position today.

We advise startups to tell investors there are several different routes they could take depending on how much they raised. As little as $50k could pay for food and rent for the founders for a year. A couple hundred thousand would let them get office space and hire some smart people they know from school. A couple million would let them really blow this thing out. The message (and not just the message, but the fact) should be: we're going to succeed no matter what. Raising more money just lets us do it faster.

If you're raising an angel round, the size of the round can even change on the fly. In fact, it's just as well to make the round small initially, then expand as needed, rather than trying to raise a large round and risk losing the investors you already have if you can't raise the full amount. You may even want to do a "rolling close," where the round has no predetermined size, but instead you sell stock to investors one at a time as they say yes. That helps break deadlocks, because you can start as soon as the first one is ready to buy. [5]

5. Be independent.

A startup with a couple founders in their early twenties can have expenses so low that they could be profitable on as little as $2000 per month. That's negligible as corporate revenues go, but the effect on your morale and your bargaining position is anything but. At YC we use the phrase "ramen profitable" to describe the situation where you're making just enough to pay your living expenses. Once you cross into ramen profitable, everything changes. You may still need investment to make it big, but you don't need it this month.

You can't plan when you start a startup how long it will take to become profitable. But if you find yourself in a position where a little more effort expended on sales would carry you over the threshold of ramen profitable, do it.

Investors like it when you're ramen profitable. It shows you've thought about making money, instead of just working on amusing technical problems; it shows you have the discipline to keep your expenses low; but above all, it means you don't need them.

There is nothing investors like more than a startup that seems like it's going to succeed even without them. Investors like it when they can help a startup, but they don't like startups that would die without that help.

At YC we spend a lot of time trying to predict how the startups we've funded will do, because we're trying to learn how to pick winners. We've now watched the trajectories of so many startups that we're getting better at predicting them. And when we're talking about startups we think are likely to succeed, what we find ourselves saying is things like "Oh, those guys can take care of themselves. They'll be fine." Not "those guys are really smart" or "those guys are working on a great idea." [6] When we predict good outcomes for startups, the qualities that come up in the supporting arguments are toughness, adaptability, determination. Which means to the extent we're correct, those are the qualities you need to win.

Investors know this, at least unconsciously. The reason they like it when you don't need them is not simply that they like what they can't have, but because that quality is what makes founders succeed.

Sam Altman has it. You could parachute him into an island full of cannibals and come back in 5 years and he'd be the king. If you're Sam Altman, you don't have to be profitable to convey to investors that you'll succeed with or without them. (He wasn't, and he did.) Not everyone has Sam's deal-making ability. I myself don't. But if you don't, you can let the numbers speak for you.

6. Don't take rejection personally.

Getting rejected by investors can make you start to doubt yourself. After all, they're more experienced than you. If they think your startup is lame, aren't they probably right?

Maybe, maybe not. The way to handle rejection is with precision. You shouldn't simply ignore rejection. It might mean something. But you shouldn't automatically get demoralized either.

To understand what rejection means, you have to understand first of all how common it is. Statistically, the average VC is a rejection machine. David Hornik, a partner at August, told me:

The numbers for me ended up being something like 500 to 800 plans received and read, somewhere between 50 and 100 initial 1 hour meetings held, about 20 companies that I got interested in, about 5 that I got serious about and did a bunch of work, 1 to 2 deals done in a year. So the odds are against you. You may be a great entrepreneur, working on interesting stuff, etc. but it is still incredibly unlikely that you get funded.

This is less true with angels, but VCs reject practically everyone. The structure of their business means a partner does at most 2 new investments a year, no matter how many good startups approach him.

In addition to the odds being terrible, the average investor is, as I mentioned, a pretty bad judge of startups. It's harder to judge startups than most other things, because great startup ideas tend to seem wrong. A good startup idea has to be not just good but novel. And to be both good and novel, an idea probably has to seem bad to most people, or someone would already be doing it and it wouldn't be novel.

That makes judging startups harder than most other things one judges. You have to be an intellectual contrarian to be a good startup investor. That's a problem for VCs, most of whom are not particularly imaginative. VCs are mostly money guys, not people who make things. [7] Angels are better at appreciating novel ideas, because most were founders themselves.

So when you get a rejection, use the data that's in it, and not what's not. If an investor gives you specific reasons for not investing, look at your startup and ask if they're right. If they're real problems, fix them. But don't just take their word for it. You're supposed to be the domain expert; you have to decide.

Though a rejection doesn't necessarily tell you anything about your startup, it does suggest your pitch could be improved. Figure out what's not working and change it. Don't just think "investors are stupid." Often they are, but figure out precisely where you lose them.

Don't let rejections pile up as a depressing, undifferentiated heap. Sort them and analyze them, and then instead of thinking "no one likes us," you'll know precisely how big a problem you have, and what to do about it.

7. Be able to downshift into consulting (if appropriate).

Consulting, as I mentioned, is a dangerous way to finance a startup. But it's better than dying. It's a bit like anaerobic respiration: not the optimum solution for the long term, but it can save you from an immediate threat. If you're having trouble raising money from investors at all, it could save you to be able to shift toward consulting.

This works better for some startups than others. It wouldn't have been a natural fit for, say, Google, but if your company was making software for building web sites, you could degrade fairly gracefully into consulting by building sites for clients with it.

So long as you were careful not to get sucked permanently into consulting, this could even have advantages. You'd understand your users well if you were using the software for them. Plus as a consulting company you might be able to get big-name users using your software that you wouldn't have gotten as a product company.

At Viaweb we were forced to operate like a consulting company initially, because we were so desperate for users that we'd offer to build merchants' sites for them if they'd sign up. But we never charged for such work, because we didn't want them to start treating us like actual consultants, and calling us every time they wanted something changed on their site. We knew we had to stay a product company, because only that scales.

8. Avoid inexperienced investors.

Though novice investors seem unthreatening they can be the most dangerous sort, because they're so nervous. Especially in proportion to the amount they invest. Raising $20,000 from a first-time angel investor can be as much work as raising $2 million from a VC fund.

Their lawyers are generally inexperienced too. But while the investors can admit they don't know what they're doing, their lawyers can't. One YC startup negotiated terms for a tiny round with an angel, only to receive a 70-page agreement from his lawyer. And since the lawyer could never admit, in front of his client, that he'd screwed up, he instead had to insist on retaining all the draconian terms in it, so the deal fell through.

Of course, someone has to take money from novice investors, or there would never be any experienced ones. But if you do, either (a) drive the process yourself, including supplying the paperwork, or (b) use them only to fill up a larger round led by someone else.

9. Know where you stand.

The most dangerous thing about investors is their indecisiveness. The worst case scenario is the long no, the no that comes after months of meetings. Rejections from investors are like design flaws: inevitable, but much less costly if you discover them early.

So while you're talking to investors, constantly look for signs of where you stand. How likely are they to offer you a term sheet? What do they have to be convinced of first? You shouldn't necessarily always be asking these questions outright—that could get annoying—but you should always be collecting data about them.

Investors tend to resist committing except to the extent you push them to. It's in their interest to collect the maximum amount of information while making the minimum number of decisions. The best way to force them to act is, of course, competing investors. But you can also apply some force by focusing the discussion: by asking what specific questions they need answered to make up their minds, and then answering them. If you get through several obstacles and they keep raising new ones, assume that ultimately they're going to flake.

You have to be disciplined when collecting data about investors' intentions. Otherwise their desire to lead you on will combine with your own desire to be led on to produce completely inaccurate impressions.

Use the data to weight your strategy. You'll probably be talking to several investors. Focus on the ones that are most likely to say yes. The value of a potential investor is a combination of how good it would be if they said yes, and how likely they are to say it. Put the most weight on the second factor. Partly because the most important quality in an investor is simply investing. But also because, as I mentioned, the biggest factor in investors' opinion of you is other investors' opinion of you. If you're talking to several investors and you manage to get one over the threshold of saying yes, it will make the others much more interested. So you're not sacrificing the lukewarm investors if you focus on the hot ones; convincing the hot investors is the best way to convince the lukewarm ones.

Future

I'm hopeful things won't always be so awkward. I hope that as startups get cheaper and the number of investors increases, raising money will become, if not easy, at least straightforward.

In the meantime, the brokenness of the funding process offers a big opportunity. Most investors have no idea how dangerous they are. They'd be surprised to hear that raising money from them is something that has to be treated as a threat to a company's survival. They just think they need a little more information to make up their minds. They don't get that there are 10 other investors who also want a little more information, and that the process of talking to them all can bring a startup to a standstill for months.

Because investors don't understand the cost of dealing with them, they don't realize how much room there is for a potential competitor to undercut them. I know from my own experience how much faster investors could decide, because we've brought our own time down to 20 minutes (5 minutes of reading an application plus a 10 minute interview plus 5 minutes of discussion). If you were investing more money you'd want to take longer, of course. But if we can decide in 20 minutes, should it take anyone longer than a couple days?

Opportunities like this don't sit unexploited forever, even in an industry as conservative as venture capital. So either existing investors will start to make up their minds faster, or new investors will emerge who do.

In the meantime founders have to treat raising money as a dangerous process. Fortunately, I can fix the biggest danger right here. The biggest danger is surprise. It's that startups will underestimate the difficulty of raising money—that they'll cruise through all the initial steps, but when they turn to raising money they'll find it surprisingly hard, get demoralized, and give up. So I'm telling you in advance: raising money is hard.

Notes

[1] When investors can't make up their minds, they sometimes describe it as if it were a property of the startup. "You're too early for us," they sometimes say. But which of them, if they were taken back in a time machine to the hour Google was founded, wouldn't offer to invest at any valuation the founders chose? An hour old is not too early if it's the right startup. What "you're too early" really means is "we can't figure out yet whether you'll succeed."

[2] Investors influence one another both directly and indirectly. They influence one another directly through the "buzz" that surrounds a hot startup. But they also influence one another indirectly through the founders. When a lot of investors are interested in you, it increases your confidence in a way that makes you much more attractive to investors.

No VC will admit they're influenced by buzz. Some genuinely aren't. But there are few who can say they're not influenced by confidence.

[3] One VC who read this essay wrote:

"We try to avoid companies that got bootstrapped with consulting. It creates very bad behaviors/instincts that are hard to erase from a company's culture."

[4] The optimal way to answer the first question is to say that it would be improper to name names, while simultaneously implying that you're talking to a bunch of other VCs who are all about to give you term sheets. If you're the sort of person who understands how to do that, go ahead. If not, don't even try. Nothing annoys VCs more than clumsy efforts to manipulate them.

[5] The disadvantage of expanding a round on the fly is that the valuation is fixed at the start, so if you get a sudden rush of interest, you may have to decide between turning some investors away and selling more of the company than you meant to. That's a good problem to have, however.

[6] I wouldn't say that intelligence doesn't matter in startups. We're only comparing YC startups, who've already made it over a certain threshold.

[7] But not all are. Though most VCs are suits at heart, the most successful ones tend not to be. Oddly enough, the best VCs tend to be the least VC-like.

The Venture Capital Squeeze

November 2005

In the next few years, venture capital funds will find themselves squeezed from four directions. They're already stuck with a seller's market, because of the huge amounts they raised at the end of the Bubble and still haven't invested. This by itself is not the end of the world. In fact, it's just a more extreme version of the norm in the VC business: too much money chasing too few deals.

Unfortunately, those few deals now want less and less money, because it's getting so cheap to start a startup. The four causes: open source, which makes software free; Moore's law, which makes hardware geometrically closer to free; the Web, which makes promotion free if you're good; and better languages, which make development a lot cheaper.

When we started our startup in 1995, the first three were our biggest expenses. We had to pay $5000 for the Netscape Commerce Server, the only software that then supported secure http connections. We paid $3000 for a server with a 90 MHz processor and 32 meg of memory. And we paid a PR firm about $30,000 to promote our launch.

Now you could get all three for nothing. You can get the software for free; people throw away computers more powerful than our first server; and if you make something good you can generate ten times as much traffic by word of mouth online than our first PR firm got through the print media.

And of course another big change for the average startup is that programming languages have improved-- or rather, the median language has. At most startups ten years ago, software development meant ten programmers writing code in C++. Now the same work might be done by one or two using Python or Ruby.

During the Bubble, a lot of people predicted that startups would outsource their development to India. I think a better model for the future is David Heinemeier Hansson, who outsourced his development to a more powerful language instead. A lot of well-known applications are now, like BaseCamp, written by just one programmer. And one guy is more than 10x cheaper than ten, because (a) he won't waste any time in meetings, and (b) since he's probably a founder, he can pay himself nothing.

Because starting a startup is so cheap, venture capitalists now often want to give startups more money than the startups want to take. VCs like to invest several million at a time. But as one VC told me after a startup he funded would only take about half a million, "I don't know what we're going to do. Maybe we'll just have to give some of it back." Meaning give some of the fund back to the institutional investors who supplied it, because it wasn't going to be possible to invest it all.

Into this already bad situation comes the third problem: Sarbanes-Oxley. Sarbanes-Oxley is a law, passed after the Bubble, that drastically increases the regulatory burden on public companies. And in addition to the cost of compliance, which is at least two million dollars a year, the law introduces frightening legal exposure for corporate officers. An experienced CFO I know said flatly: "I would not want to be CFO of a public company now."

You might think that responsible corporate governance is an area where you can't go too far. But you can go too far in any law, and this remark convinced me that Sarbanes-Oxley must have. This CFO is both the smartest and the most upstanding money guy I know. If Sarbanes-Oxley deters people like him from being CFOs of public companies, that's proof enough that it's broken.

Largely because of Sarbanes-Oxley, few startups go public now. For all practical purposes, succeeding now equals getting bought. Which means VCs are now in the business of finding promising little 2-3 man startups and pumping them up into companies that cost $100 million to acquire. They didn't mean to be in this business; it's just what their business has evolved into.

Hence the fourth problem: the acquirers have begun to realize they can buy wholesale. Why should they wait for VCs to make the startups they want more expensive? Most of what the VCs add, acquirers don't want anyway. The acquirers already have brand recognition and HR departments. What they really want is the software and the developers, and that's what the startup is in the early phase: concentrated software and developers.

Google, typically, seems to have been the first to figure this out. "Bring us your startups early," said Google's speaker at the Startup School. They're quite explicit about it: they like to acquire startups at just the point where they would do a Series A round. (The Series A round is the first round of real VC funding; it usually happens in the first year.) It is a brilliant strategy, and one that other big technology companies will no doubt try to duplicate. Unless they want to have still more of their lunch eaten by Google.

Of course, Google has an advantage in buying startups: a lot of the people there are rich, or expect to be when their options vest. Ordinary employees find it very hard to recommend an acquisition; it's just too annoying to see a bunch of twenty year olds get rich when you're still working for salary. Even if it's the right thing for your company to do.

The Solution(s)

Bad as things look now, there is a way for VCs to save themselves. They need to do two things, one of which won't surprise them, and another that will seem an anathema.

Let's start with the obvious one: lobby to get Sarbanes-Oxley loosened. This law was created to prevent future Enrons, not to destroy the IPO market. Since the IPO market was practically dead when it passed, few saw what bad effects it would have. But now that technology has recovered from the last bust, we can see clearly what a bottleneck Sarbanes-Oxley has become.

Startups are fragile plants—seedlings, in fact. These seedlings are worth protecting, because they grow into the trees of the economy. Much of the economy's growth is their growth. I think most politicians realize that. But they don't realize just how fragile startups are, and how easily they can become collateral damage of laws meant to fix some other problem.

Still more dangerously, when you destroy startups, they make very little noise. If you step on the toes of the coal industry, you'll hear about it. But if you inadvertantly squash the startup industry, all that happens is that the founders of the next Google stay in grad school instead of starting a company.

My second suggestion will seem shocking to VCs: let founders cash out partially in the Series A round. At the moment, when VCs invest in a startup, all the stock they get is newly issued and all the money goes to the company. They could buy some stock directly from the founders as well.

Most VCs have an almost religious rule against doing this. They don't want founders to get a penny till the company is sold or goes public. VCs are obsessed with control, and they worry that they'll have less leverage over the founders if the founders have any money.

This is a dumb plan. In fact, letting the founders sell a little stock early would generally be better for the company, because it would cause the founders' attitudes toward risk to be aligned with the VCs'. As things currently work, their attitudes toward risk tend to be diametrically opposed: the founders, who have nothing, would prefer a 100% chance of $1 million to a 20% chance of $10 million, while the VCs can afford to be "rational" and prefer the latter.

Whatever they say, the reason founders are selling their companies early instead of doing Series A rounds is that they get paid up front. That first million is just worth so much more than the subsequent ones. If founders could sell a little stock early, they'd be happy to take VC money and bet the rest on a bigger outcome.

So why not let the founders have that first million, or at least half million? The VCs would get same number of shares for the money. So what if some of the money would go to the founders instead of the company?

Some VCs will say this is unthinkable—that they want all their money to be put to work growing the company. But the fact is, the huge size of current VC investments is dictated by the structure of VC funds, not the needs of startups. Often as not these large investments go to work destroying the company rather than growing it.

The angel investors who funded our startup let the founders sell some stock directly to them, and it was a good deal for everyone. The angels made a huge return on that investment, so they're happy. And for us founders it blunted the terrifying all-or-nothingness of a startup, which in its raw form is more a distraction than a motivator.

If VCs are frightened at the idea of letting founders partially cash out, let me tell them something still more frightening: you are now competing directly with Google.

The Other Road Ahead

September 2001

(This article explains why much of the next generation of software may be server-based, what that will mean for programmers, and why this new kind of software is a great opportunity for startups. It's derived from a talk at BBN Labs.)

In the summer of 1995, my friend Robert Morris and I decided to start a startup. The PR campaign leading up to Netscape's IPO was running full blast then, and there was a lot of talk in the press about online commerce. At the time there might have been thirty actual stores on the Web, all made by hand. If there were going to be a lot of online stores, there would need to be software for making them, so we decided to write some.

For the first week or so we intended to make this an ordinary desktop application. Then one day we had the idea of making the software run on our Web server, using the browser as an interface. We tried rewriting the software to work over the Web, and it was clear that this was the way to go. If we wrote our software to run on the server, it would be a lot easier for the users and for us as well.

This turned out to be a good plan. Now, as Yahoo Store, this software is the most popular online store builder, with about 14,000 users.

When we started Viaweb, hardly anyone understood what we meant when we said that the software ran on the server. It was not until Hotmail was launched a year later that people started to get it. Now everyone knows that this is a valid approach. There is a name now for what we were: an Application Service Provider, or ASP.

I think that a lot of the next generation of software will be written on this model. Even Microsoft, who have the most to lose, seem to see the inevitablity of moving some things off the desktop. If software moves off the desktop and onto servers, it will mean a very different world for developers. This article describes the surprising things we saw, as some of the first visitors to this new world. To the extent software does move onto servers, what I'm describing here is the future.

The Next Thing?

When we look back on the desktop software era, I think we'll marvel at the inconveniences people put up with, just as we marvel now at what early car owners put up with. For the first twenty or thirty years, you had to be a car expert to own a car. But cars were such a big win that lots of people who weren't car experts wanted to have them as well.

Computers are in this phase now. When you own a desktop computer, you end up learning a lot more than you wanted to know about what's happening inside it. But more than half the households in the US own one. My mother has a computer that she uses for email and for keeping accounts. About a year ago she was alarmed to receive a letter from Apple, offering her a discount on a new version of the operating system. There's something wrong when a sixty-five year old woman who wants to use a computer for email and accounts has to think about installing new operating systems. Ordinary users shouldn't even know the words "operating system," much less "device driver" or "patch."

There is now another way to deliver software that will save users from becoming system administrators. Web-based applications are programs that run on Web servers and use Web pages as the user interface. For the average user this new kind of software will be easier, cheaper, more mobile, more reliable, and often more powerful than desktop software.

With Web-based software, most users won't have to think about anything except the applications they use. All the messy, changing stuff will be sitting on a server somewhere, maintained by the kind of people who are good at that kind of thing. And so you won't ordinarily need a computer, per se, to use software. All you'll need will be something with a keyboard, a screen, and a Web browser. Maybe it will have wireless Internet access. Maybe it will also be your cell phone. Whatever it is, it will be consumer electronics: something that costs about $200, and that people choose mostly based on how the case looks. You'll pay more for Internet services than you do for the hardware, just as you do now with telephones. [1]

It will take about a tenth of a second for a click to get to the server and back, so users of heavily interactive software, like Photoshop, will still want to have the computations happening on the desktop. But if you look at the kind of things most people use computers for, a tenth of a second latency would not be a problem. My mother doesn't really need a desktop computer, and there are a lot of people like her.

The Win for Users

Near my house there is a car with a bumper sticker that reads "death before inconvenience." Most people, most of the time, will take whatever choice requires least work. If Web-based software wins, it will be because it's more convenient. And it looks as if it will be, for users and developers both.

To use a purely Web-based application, all you need is a browser connected to the Internet. So you can use a Web-based application anywhere. When you install software on your desktop computer, you can only use it on that computer. Worse still, your files are trapped on that computer. The inconvenience of this model becomes more and more evident as people get used to networks.

The thin end of the wedge here was Web-based email. Millions of people now realize that you should have access to email messages no matter where you are. And if you can see your email, why not your calendar? If you can discuss a

document with your colleagues, why can't you edit it? Why should any of your data be trapped on some computer sitting on a faraway desk?

The whole idea of "your computer" is going away, and being replaced with "your data." You should be able to get at your data from any computer. Or rather, any client, and a client doesn't have to be a computer.

Clients shouldn't store data; they should be like telephones. In fact they may become telephones, or vice versa. And as clients get smaller, you have another reason not to keep your data on them: something you carry around with you can be lost or stolen. Leaving your PDA in a taxi is like a disk crash, except that your data is handed to someone else instead of being vaporized.

With purely Web-based software, neither your data nor the applications are kept on the client. So you don't have to install anything to use it. And when there's no installation, you don't have to worry about installation going wrong. There can't be incompatibilities between the application and your operating system, because the software doesn't run on your operating system.

Because it needs no installation, it will be easy, and common, to try Web-based software before you "buy" it. You should expect to be able to test-drive any Web-based application for free, just by going to the site where it's offered. At Viaweb our whole site was like a big arrow pointing users to the test drive.

After trying the demo, signing up for the service should require nothing more than filling out a brief form (the briefer the better). And that should be the last work the user has to do. With Web-based software, you should get new releases without paying extra, or doing any work, or possibly even knowing about it.

Upgrades won't be the big shocks they are now. Over time applications will quietly grow more powerful. This will take some effort on the part of the developers. They will have to design software so that it can be updated without confusing the users. That's a new problem, but there are ways to solve it.

With Web-based applications, everyone uses the same version, and bugs can be fixed as soon as they're discovered. So Web-based software should have far fewer bugs than desktop software. At Viaweb, I doubt we ever had ten known bugs at any one time. That's orders of magnitude better than desktop software.

Web-based applications can be used by several people at the same time. This is an obvious win for collaborative applications, but I bet users will start to want this in most applications once they realize it's possible. It will often be useful to let two people edit the same document, for example. Viaweb let multiple users edit a site simultaneously, more because that was the right way to write the software than because we expected users to want to, but it turned out that many did.

When you use a Web-based application, your data will be safer. Disk crashes won't be a thing of the past, but users won't hear about them anymore. They'll happen within server farms. And companies offering Web-based

applications will actually do backups-- not only because they'll have real system administrators worrying about such things, but because an ASP that does lose people's data will be in big, big trouble. When people lose their own data in a disk crash, they can't get that mad, because they only have themselves to be mad at. When a company loses their data for them, they'll get a lot madder.

Finally, Web-based software should be less vulnerable to viruses. If the client doesn't run anything except a browser, there's less chance of running viruses, and no data locally to damage. And a program that attacked the servers themselves should find them very well defended. [2]

For users, Web-based software will be less stressful. I think if you looked inside the average Windows user you'd find a huge and pretty much untapped desire for software meeting that description. Unleashed, it could be a powerful force.

City of Code

To developers, the most conspicuous difference between Web-based and desktop software is that a Web-based application is not a single piece of code. It will be a collection of programs of different types rather than a single big binary. And so designing Web-based software is like desiging a city rather than a building: as well as buildings you need roads, street signs, utilities, police and fire departments, and plans for both growth and various kinds of disasters.

At Viaweb, software included fairly big applications that users talked to directly, programs that those programs used, programs that ran constantly in the background looking for problems, programs that tried to restart things if they broke, programs that ran occasionally to compile statistics or build indexes for searches, programs we ran explicitly to garbage-collect resources or to move or restore data, programs that pretended to be users (to measure performance or expose bugs), programs for diagnosing network troubles, programs for doing backups, interfaces to outside services, software that drove an impressive collection of dials displaying real-time server statistics (a hit with visitors, but indispensable for us too), modifications (including bug fixes) to open-source software, and a great many configuration files and settings. Trevor Blackwell wrote a spectacular program for moving stores to new servers across the country, without shutting them down, after we were bought by Yahoo. Programs paged us, sent faxes and email to users, conducted transactions with credit card processors, and talked to one another through sockets, pipes, http requests, ssh, udp packets, shared memory, and files. Some of Viaweb even consisted of the absence of programs, since one of the keys to Unix security is not to run unnecessary utilities that people might use to break into your servers.

It did not end with software. We spent a lot of time thinking about server configurations. We built the servers ourselves, from components-- partly to save money, and partly to get exactly what we wanted. We had to think about whether our upstream ISP had fast enough connections to all the backbones. We serially dated RAID suppliers.

But hardware is not just something to worry about. When you control it you can do more for users. With a desktop application, you can specify certain minimum hardware, but you can't add more. If you administer the servers, you can in one step enable all your users to page people, or send faxes, or send commands by phone, or process credit

cards, etc, just by installing the relevant hardware. We always looked for new ways to add features with hardware, not just because it pleased users, but also as a way to distinguish ourselves from competitors who (either because they sold desktop software, or resold Web-based applications through ISPs) didn't have direct control over the hardware.

Because the software in a Web-based application will be a collection of programs rather than a single binary, it can be written in any number of different languages. When you're writing desktop software, you're practically forced to write the application in the same language as the underlying operating system-- meaning C and C++. And so these languages (especially among nontechnical people like managers and VCs) got to be considered as the languages for "serious" software development. But that was just an artifact of the way desktop software had to be delivered. For server-based software you can use any language you want. [3] Today a lot of the top hackers are using languages far removed from C and C++: Perl, Python, and even Lisp.

With server-based software, no one can tell you what language to use, because you control the whole system, right down to the hardware. Different languages are good for different tasks. You can use whichever is best for each. And when you have competitors, "you can" means "you must" (we'll return to this later), because if you don't take advantage of this possibility, your competitors will.

Most of our competitors used C and C++, and this made their software visibly inferior because (among other things), they had no way around the statelessness of CGI scripts. If you were going to change something, all the changes had to happen on one page, with an Update button at the bottom. As I've written elsewhere, by using Lisp, which many people still consider a research language, we could make the Viaweb editor behave more like desktop software.

Releases

One of the most important changes in this new world is the way you do releases. In the desktop software business, doing a release is a huge trauma, in which the whole company sweats and strains to push out a single, giant piece of code. Obvious comparisons suggest themselves, both to the process and the resulting product.

With server-based software, you can make changes almost as you would in a program you were writing for yourself. You release software as a series of incremental changes instead of an occasional big explosion. A typical desktop software company might do one or two releases a year. At Viaweb we often did three to five releases a day.

When you switch to this new model, you realize how much software development is affected by the way it is released. Many of the nastiest problems you see in the desktop software business are due to catastrophic nature of releases.

When you release only one new version a year, you tend to deal with bugs wholesale. Some time before the release date you assemble a new version in which half the code has been torn out and replaced, introducing countless bugs. Then a squad of QA people step in and start counting them, and the programmers work down the list, fixing them. They do not generally get to the end of the list, and indeed, no one is sure where the end is. It's like fishing rubble

out of a pond. You never really know what's happening inside the software. At best you end up with a statistical sort of correctness.

With server-based software, most of the change is small and incremental. That in itself is less likely to introduce bugs. It also means you know what to test most carefully when you're about to release software: the last thing you changed. You end up with a much firmer grip on the code. As a general rule, you do know what's happening inside it. You don't have the source code memorized, of course, but when you read the source you do it like a pilot scanning the instrument panel, not like a detective trying to unravel some mystery.

Desktop software breeds a certain fatalism about bugs. You know that you're shipping something loaded with bugs, and you've even set up mechanisms to compensate for it (e.g. patch releases). So why worry about a few more? Soon you're releasing whole features you know are broken. Apple did this earlier this year. They felt under pressure to release their new OS, whose release date had already slipped four times, but some of the software (support for CDs and DVDs) wasn't ready. The solution? They released the OS without the unfinished parts, and users will have to install them later.

With Web-based software, you never have to release software before it works, and you can release it as soon as it does work.

The industry veteran may be thinking, it's a fine-sounding idea to say that you never have to release software before it works, but what happens when you've promised to deliver a new version of your software by a certain date? With Web-based software, you wouldn't make such a promise, because there are no versions. Your software changes gradually and continuously. Some changes might be bigger than others, but the idea of versions just doesn't naturally fit onto Web-based software.

If anyone remembers Viaweb this might sound odd, because we were always announcing new versions. This was done entirely for PR purposes. The trade press, we learned, thinks in version numbers. They will give you major coverage for a major release, meaning a new first digit on the version number, and generally a paragraph at most for a point release, meaning a new digit after the decimal point.

Some of our competitors were offering desktop software and actually had version numbers. And for these releases, the mere fact of which seemed to us evidence of their backwardness, they would get all kinds of publicity. We didn't want to miss out, so we started giving version numbers to our software too. When we wanted some publicity, we'd make a list of all the features we'd added since the last "release," stick a new version number on the software, and issue a press release saying that the new version was available immediately. Amazingly, no one ever called us on it.

By the time we were bought, we had done this three times, so we were on Version 4. Version 4.1 if I remember correctly. After Viaweb became Yahoo Store, there was no longer such a desperate need for publicity, so although the software continued to evolve, the whole idea of version numbers was quietly dropped.

Bugs

The other major technical advantage of Web-based software is that you can reproduce most bugs. You have the users' data right there on your disk. If someone breaks your software, you don't have to try to guess what's going on, as you would with desktop software: you should be able to reproduce the error while they're on the phone with you. You might even know about it already, if you have code for noticing errors built into your application.

Web-based software gets used round the clock, so everything you do is immediately put through the wringer. Bugs turn up quickly.

Software companies are sometimes accused of letting the users debug their software. And that is just what I'm advocating. For Web-based software it's actually a good plan, because the bugs are fewer and transient. When you release software gradually you get far fewer bugs to start with. And when you can reproduce errors and release changes instantly, you can find and fix most bugs as soon as they appear. We never had enough bugs at any one time to bother with a formal bug-tracking system.

You should test changes before you release them, of course, so no major bugs should get released. Those few that inevitably slip through will involve borderline cases and will only affect the few users that encounter them before someone calls in to complain. As long as you fix bugs right away, the net effect, for the average user, is far fewer bugs. I doubt the average Viaweb user ever saw a bug.

Fixing fresh bugs is easier than fixing old ones. It's usually fairly quick to find a bug in code you just wrote. When it turns up you often know what's wrong before you even look at the source, because you were already worrying about it subconsciously. Fixing a bug in something you wrote six months ago (the average case if you release once a year) is a lot more work. And since you don't understand the code as well, you're more likely to fix it in an ugly way, or even introduce more bugs. [4]

When you catch bugs early, you also get fewer compound bugs. Compound bugs are two separate bugs that interact: you trip going downstairs, and when you reach for the handrail it comes off in your hand. In software this kind of bug is the hardest to find, and also tends to have the worst consequences. [5] The traditional "break everything and then filter out the bugs" approach inherently yields a lot of compound bugs. And software that's released in a series of small changes inherently tends not to. The floors are constantly being swept clean of any loose objects that might later get stuck in something.

It helps if you use a technique called functional programming. Functional programming means avoiding side-effects. It's something you're more likely to see in research papers than commercial software, but for Web-based applications it turns out to be really useful. It's hard to write entire programs as purely functional code, but you can write substantial chunks this way. It makes those parts of your software easier to test, because they have no state, and that is very convenient in a situation where you are constantly making and testing small modifications. I wrote much of Viaweb's editor in this style, and we made our scripting language, RTML, a purely functional language.

People from the desktop software business will find this hard to credit, but at Viaweb bugs became almost a game. Since most released bugs involved borderline cases, the users who encountered them were likely to be advanced users, pushing the envelope. Advanced users are more forgiving about bugs, especially since you probably introduced them in the course of adding some feature they were asking for. In fact, because bugs were rare and you had to be doing sophisticated things to see them, advanced users were often proud to catch one. They would call support in a spirit more of triumph than anger, as if they had scored points off us.

Support

When you can reproduce errors, it changes your approach to customer support. At most software companies, support is offered as a way to make customers feel better. They're either calling you about a known bug, or they're just doing something wrong and you have to figure out what. In either case there's not much you can learn from them. And so you tend to view support calls as a pain in the ass that you want to isolate from your developers as much as possible.

This was not how things worked at Viaweb. At Viaweb, support was free, because we wanted to hear from customers. If someone had a problem, we wanted to know about it right away so that we could reproduce the error and release a fix.

So at Viaweb the developers were always in close contact with support. The customer support people were about thirty feet away from the programmers, and knew that they could always interrupt anything with a report of a genuine bug. We would leave a board meeting to fix a serious bug.

Our approach to support made everyone happier. The customers were delighted. Just imagine how it would feel to call a support line and be treated as someone bringing important news. The customer support people liked it because it meant they could help the users, instead of reading scripts to them. And the programmers liked it because they could reproduce bugs instead of just hearing vague second-hand reports about them.

Our policy of fixing bugs on the fly changed the relationship between customer support people and hackers. At most software companies, support people are underpaid human shields, and hackers are little copies of God the Father, creators of the world. Whatever the procedure for reporting bugs, it is likely to be one-directional: support people who hear about bugs fill out some form that eventually gets passed on (possibly via QA) to programmers, who put it on their list of things to do. It was very different at Viaweb. Within a minute of hearing about a bug from a customer, the support people could be standing next to a programmer hearing him say "Shit, you're right, it's a bug." It delighted the support people to hear that "you're right" from the hackers. They used to bring us bugs with the same expectant air as a cat bringing you a mouse it has just killed. It also made them more careful in judging the seriousness of a bug, because now their honor was on the line.

After we were bought by Yahoo, the customer support people were moved far away from the programmers. It was only then that we realized that they were effectively QA and to some extent marketing as well. In addition to catching bugs, they were the keepers of the knowledge of vaguer, buglike things, like features that confused users.

[6] They were also a kind of proxy focus group; we could ask them which of two new features users wanted more, and they were always right.

Morale

Being able to release software immediately is a big motivator. Often as I was walking to work I would think of some change I wanted to make to the software, and do it that day. This worked for bigger features as well. Even if something was going to take two weeks to write (few projects took longer), I knew I could see the effect in the software as soon as it was done.

If I'd had to wait a year for the next release, I would have shelved most of these ideas, for a while at least. The thing about ideas, though, is that they lead to more ideas. Have you ever noticed that when you sit down to write something, half the ideas that end up in it are ones you thought of while writing it? The same thing happens with software. Working to implement one idea gives you more ideas. So shelving an idea costs you not only that delay in implementing it, but also all the ideas that implementing it would have led to. In fact, shelving an idea probably even inhibits new ideas: as you start to think of some new feature, you catch sight of the shelf and think "but I already have a lot of new things I want to do for the next release."

What big companies do instead of implementing features is plan them. At Viaweb we sometimes ran into trouble on this account. Investors and analysts would ask us what we had planned for the future. The truthful answer would have been, we didn't have any plans. We had general ideas about things we wanted to improve, but if we knew how we would have done it already. What were we going to do in the next six months? Whatever looked like the biggest win. I don't know if I ever dared give this answer, but that was the truth. Plans are just another word for ideas on the shelf. When we thought of good ideas, we implemented them.

At Viaweb, as at many software companies, most code had one definite owner. But when you owned something you really owned it: no one except the owner of a piece of software had to approve (or even know about) a release. There was no protection against breakage except the fear of looking like an idiot to one's peers, and that was more than enough. I may have given the impression that we just blithely plowed forward writing code. We did go fast, but we thought very carefully before we released software onto those servers. And paying attention is more important to reliability than moving slowly. Because he pays close attention, a Navy pilot can land a 40,000 lb. aircraft at 140 miles per hour on a pitching carrier deck, at night, more safely than the average teenager can cut a bagel.

This way of writing software is a double-edged sword of course. It works a lot better for a small team of good, trusted programmers than it would for a big company of mediocre ones, where bad ideas are caught by committees instead of the people that had them.

Brooks in Reverse

Fortunately, Web-based software does require fewer programmers. I once worked for a medium-sized desktop software company that had over 100 people working in engineering as a whole. Only 13 of these were in product

development. All the rest were working on releases, ports, and so on. With Web-based software, all you need (at most) are the 13 people, because there are no releases, ports, and so on.

Viaweb was written by just three people. [7] I was always under pressure to hire more, because we wanted to get bought, and we knew that buyers would have a hard time paying a high price for a company with only three programmers. (Solution: we hired more, but created new projects for them.)

When you can write software with fewer programmers, it saves you more than money. As Fred Brooks pointed out in The Mythical Man-Month, adding people to a project tends to slow it down. The number of possible connections between developers grows exponentially with the size of the group. The larger the group, the more time they'll spend in meetings negotiating how their software will work together, and the more bugs they'll get from unforeseen interactions. Fortunately, this process also works in reverse: as groups get smaller, software development gets exponentially more efficient. I can't remember the programmers at Viaweb ever having an actual meeting. We never had more to say at any one time than we could say as we were walking to lunch.

If there is a downside here, it is that all the programmers have to be to some degree system administrators as well. When you're hosting software, someone has to be watching the servers, and in practice the only people who can do this properly are the ones who wrote the software. At Viaweb our system had so many components and changed so frequently that there was no definite border between software and infrastructure. Arbitrarily declaring such a border would have constrained our design choices. And so although we were constantly hoping that one day ("in a couple months") everything would be stable enough that we could hire someone whose job was just to worry about the servers, it never happened.

I don't think it could be any other way, as long as you're still actively developing the product. Web-based software is never going to be something you write, check in, and go home. It's a live thing, running on your servers right now. A bad bug might not just crash one user's process; it could crash them all. If a bug in your code corrupts some data on disk, you have to fix it. And so on. We found that you don't have to watch the servers every minute (after the first year or so), but you definitely want to keep an eye on things you've changed recently. You don't release code late at night and then go home.

Watching Users

With server-based software, you're in closer touch with your code. You can also be in closer touch with your users. Intuit is famous for introducing themselves to customers at retail stores and asking to follow them home. If you've ever watched someone use your software for the first time, you know what surprises must have awaited them.

Software should do what users think it will. But you can't have any idea what users will be thinking, believe me, until you watch them. And server-based software gives you unprecedented information about their behavior. You're not limited to small, artificial focus groups. You can see every click made by every user. You have to consider carefully what you're going to look at, because you don't want to violate users' privacy, but even the most general statistical sampling can be very useful.

When you have the users on your server, you don't have to rely on benchmarks, for example. Benchmarks are simulated users. With server-based software, you can watch actual users. To decide what to optimize, just log into a server and see what's consuming all the CPU. And you know when to stop optimizing too: we eventually got the Viaweb editor to the point where it was memory-bound rather than CPU-bound, and since there was nothing we could do to decrease the size of users' data (well, nothing easy), we knew we might as well stop there.

Efficiency matters for server-based software, because you're paying for the hardware. The number of users you can support per server is the divisor of your capital cost, so if you can make your software very efficient you can undersell competitors and still make a profit. At Viaweb we got the capital cost per user down to about $5. It would be less now, probably less than the cost of sending them the first month's bill. Hardware is free now, if your software is reasonably efficient.

Watching users can guide you in design as well as optimization. Viaweb had a scripting language called RTML that let advanced users define their own page styles. We found that RTML became a kind of suggestion box, because users only used it when the predefined page styles couldn't do what they wanted. Originally the editor put button bars across the page, for example, but after a number of users used RTML to put buttons down the left side, we made that an option (in fact the default) in the predefined page styles.

Finally, by watching users you can often tell when they're in trouble. And since the customer is always right, that's a sign of something you need to fix. At Viaweb the key to getting users was the online test drive. It was not just a series of slides built by marketing people. In our test drive, users actually used the software. It took about five minutes, and at the end of it they had built a real, working store.

The test drive was the way we got nearly all our new users. I think it will be the same for most Web-based applications. If users can get through a test drive successfully, they'll like the product. If they get confused or bored, they won't. So anything we could do to get more people through the test drive would increase our growth rate.

I studied click trails of people taking the test drive and found that at a certain step they would get confused and click on the browser's Back button. (If you try writing Web-based applications, you'll find that the Back button becomes one of your most interesting philosophical problems.) So I added a message at that point, telling users that they were nearly finished, and reminding them not to click on the Back button. Another great thing about Web-based software is that you get instant feedback from changes: the number of people completing the test drive rose immediately from 60% to 90%. And since the number of new users was a function of the number of completed test drives, our revenue growth increased by 50%, just from that change.

Money

In the early 1990s I read an article in which someone said that software was a subscription business. At first this seemed a very cynical statement. But later I realized that it reflects reality: software development is an ongoing process. I think it's cleaner if you openly charge subscription fees, instead of forcing people to keep buying and

installing new versions so that they'll keep paying you. And fortunately, subscriptions are the natural way to bill for Web-based applications.

Hosting applications is an area where companies will play a role that is not likely to be filled by freeware. Hosting applications is a lot of stress, and has real expenses. No one is going to want to do it for free.

For companies, Web-based applications are an ideal source of revenue. Instead of starting each quarter with a blank slate, you have a recurring revenue stream. Because your software evolves gradually, you don't have to worry that a new model will flop; there never need be a new model, per se, and if you do something to the software that users hate, you'll know right away. You have no trouble with uncollectable bills; if someone won't pay you can just turn off the service. And there is no possibility of piracy.

That last "advantage" may turn out to be a problem. Some amount of piracy is to the advantage of software companies. If some user really would not have bought your software at any price, you haven't lost anything if he uses a pirated copy. In fact you gain, because he is one more user helping to make your software the standard-- or who might buy a copy later, when he graduates from high school.

When they can, companies like to do something called price discrimination, which means charging each customer as much as they can afford. [8] Software is particularly suitable for price discrimination, because the marginal cost is close to zero. This is why some software costs more to run on Suns than on Intel boxes: a company that uses Suns is not interested in saving money and can safely be charged more. Piracy is effectively the lowest tier of price discrimination. I think that software companies understand this and deliberately turn a blind eye to some kinds of piracy. [9] With server-based software they are going to have to come up with some other solution.

Web-based software sells well, especially in comparison to desktop software, because it's easy to buy. You might think that people decide to buy something, and then buy it, as two separate steps. That's what I thought before Viaweb, to the extent I thought about the question at all. In fact the second step can propagate back into the first: if something is hard to buy, people will change their mind about whether they wanted it. And vice versa: you'll sell more of something when it's easy to buy. I buy more books because Amazon exists. Web-based software is just about the easiest thing in the world to buy, especially if you have just done an online demo. Users should not have to do much more than enter a credit card number. (Make them do more at your peril.)

Sometimes Web-based software is offered through ISPs acting as resellers. This is a bad idea. You have to be administering the servers, because you need to be constantly improving both hardware and software. If you give up direct control of the servers, you give up most of the advantages of developing Web-based applications.

Several of our competitors shot themselves in the foot this way-- usually, I think, because they were overrun by suits who were excited about this huge potential channel, and didn't realize that it would ruin the product they hoped to sell through it. Selling Web-based software through ISPs is like selling sushi through vending machines.

Customers

Who will the customers be? At Viaweb they were initially individuals and smaller companies, and I think this will be the rule with Web-based applications. These are the users who are ready to try new things, partly because they're more flexible, and partly because they want the lower costs of new technology.

Web-based applications will often be the best thing for big companies too (though they'll be slow to realize it). The best intranet is the Internet. If a company uses true Web-based applications, the software will work better, the servers will be better administered, and employees will have access to the system from anywhere.

The argument against this approach usually hinges on security: if access is easier for employees, it will be for bad guys too. Some larger merchants were reluctant to use Viaweb because they thought customers' credit card information would be safer on their own servers. It was not easy to make this point diplomatically, but in fact the data was almost certainly safer in our hands than theirs. Who can hire better people to manage security, a technology startup whose whole business is running servers, or a clothing retailer? Not only did we have better people worrying about security, we worried more about it. If someone broke into the clothing retailer's servers, it would affect at most one merchant, could probably be hushed up, and in the worst case might get one person fired. If someone broke into ours, it could affect thousands of merchants, would probably end up as news on CNet, and could put us out of business.

If you want to keep your money safe, do you keep it under your mattress at home, or put it in a bank? This argument applies to every aspect of server administration: not just security, but uptime, bandwidth, load management, backups, etc. Our existence depended on doing these things right. Server problems were the big no-no for us, like a dangerous toy would be for a toy maker, or a salmonella outbreak for a food processor.

A big company that uses Web-based applications is to that extent outsourcing IT. Drastic as it sounds, I think this is generally a good idea. Companies are likely to get better service this way than they would from in-house system administrators. System administrators can become cranky and unresponsive because they're not directly exposed to competitive pressure: a salesman has to deal with customers, and a developer has to deal with competitors' software, but a system administrator, like an old bachelor, has few external forces to keep him in line. [10] At Viaweb we had external forces in plenty to keep us in line. The people calling us were customers, not just co-workers. If a server got wedged, we jumped; just thinking about it gives me a jolt of adrenaline, years later.

So Web-based applications will ordinarily be the right answer for big companies too. They will be the last to realize it, however, just as they were with desktop computers. And partly for the same reason: it will be worth a lot of money to convince big companies that they need something more expensive.

There is always a tendency for rich customers to buy expensive solutions, even when cheap solutions are better, because the people offering expensive solutions can spend more to sell them. At Viaweb we were always up against this. We lost several high-end merchants to Web consulting firms who convinced them they'd be better off if they paid half a million dollars for a custom-made online store on their own server. They were, as a rule, not better off, as more than one discovered when Christmas shopping season came around and loads rose on their server. Viaweb was a lot more sophisticated than what most of these merchants got, but we couldn't afford to tell them. At $300 a

month, we couldn't afford to send a team of well-dressed and authoritative-sounding people to make presentations to customers.

A large part of what big companies pay extra for is the cost of selling expensive things to them. (If the Defense Department pays a thousand dollars for toilet seats, it's partly because it costs a lot to sell toilet seats for a thousand dollars.) And this is one reason intranet software will continue to thrive, even though it is probably a bad idea. It's simply more expensive. There is nothing you can do about this conundrum, so the best plan is to go for the smaller customers first. The rest will come in time.

Son of Server

Running software on the server is nothing new. In fact it's the old model: mainframe applications are all server-based. If server-based software is such a good idea, why did it lose last time? Why did desktop computers eclipse mainframes?

At first desktop computers didn't look like much of a threat. The first users were all hackers-- or hobbyists, as they were called then. They liked microcomputers because they were cheap. For the first time, you could have your own computer. The phrase "personal computer" is part of the language now, but when it was first used it had a deliberately audacious sound, like the phrase "personal satellite" would today.

Why did desktop computers take over? I think it was because they had better software. And I think the reason microcomputer software was better was that it could be written by small companies.

I don't think many people realize how fragile and tentative startups are in the earliest stage. Many startups begin almost by accident-- as a couple guys, either with day jobs or in school, writing a prototype of something that might, if it looks promising, turn into a company. At this larval stage, any significant obstacle will stop the startup dead in its tracks. Writing mainframe software required too much commitment up front. Development machines were expensive, and because the customers would be big companies, you'd need an impressive-looking sales force to sell it to them. Starting a startup to write mainframe software would be a much more serious undertaking than just hacking something together on your Apple II in the evenings. And so you didn't get a lot of startups writing mainframe applications.

The arrival of desktop computers inspired a lot of new software, because writing applications for them seemed an attainable goal to larval startups. Development was cheap, and the customers would be individual people that you could reach through computer stores or even by mail-order.

The application that pushed desktop computers out into the mainstream was VisiCalc, the first spreadsheet. It was written by two guys working in an attic, and yet did things no mainframe software could do. [11] VisiCalc was such an advance, in its time, that people bought Apple IIs just to run it. And this was the beginning of a trend: desktop computers won because startups wrote software for them.

It looks as if server-based software will be good this time around, because startups will write it. Computers are so cheap now that you can get started, as we did, using a desktop computer as a server. Inexpensive processors have eaten the workstation market (you rarely even hear the word now) and are most of the way through the server market; Yahoo's servers, which deal with loads as high as any on the Internet, all have the same inexpensive Intel processors that you have in your desktop machine. And once you've written the software, all you need to sell it is a Web site. Nearly all our users came direct to our site through word of mouth and references in the press. [12]

Viaweb was a typical larval startup. We were terrified of starting a company, and for the first few months comforted ourselves by treating the whole thing as an experiment that we might call off at any moment. Fortunately, there were few obstacles except technical ones. While we were writing the software, our Web server was the same desktop machine we used for development, connected to the outside world by a dialup line. Our only expenses in that phase were food and rent.

There is all the more reason for startups to write Web-based software now, because writing desktop software has become a lot less fun. If you want to write desktop software now you do it on Microsoft's terms, calling their APIs and working around their buggy OS. And if you manage to write something that takes off, you may find that you were merely doing market research for Microsoft.

If a company wants to make a platform that startups will build on, they have to make it something that hackers themselves will want to use. That means it has to be inexpensive and well-designed. The Mac was popular with hackers when it first came out, and a lot of them wrote software for it. [13] You see this less with Windows, because hackers don't use it. The kind of people who are good at writing software tend to be running Linux or FreeBSD now.

I don't think we would have started a startup to write desktop software, because desktop software has to run on Windows, and before we could write software for Windows we'd have to use it. The Web let us do an end-run around Windows, and deliver software running on Unix direct to users through the browser. That is a liberating prospect, a lot like the arrival of PCs twenty-five years ago.

Microsoft

Back when desktop computers arrived, IBM was the giant that everyone was afraid of. It's hard to imagine now, but I remember the feeling very well. Now the frightening giant is Microsoft, and I don't think they are as blind to the threat facing them as IBM was. After all, Microsoft deliberately built their business in IBM's blind spot.

I mentioned earlier that my mother doesn't really need a desktop computer. Most users probably don't. That's a problem for Microsoft, and they know it. If applications run on remote servers, no one needs Windows. What will Microsoft do? Will they be able to use their control of the desktop to prevent, or constrain, this new generation of software?

My guess is that Microsoft will develop some kind of server/desktop hybrid, where the operating system works together with servers they control. At a minimum, files will be centrally available for users who want that. I don't expect Microsoft to go all the way to the extreme of doing the computations on the server, with only a browser for a client, if they can avoid it. If you only need a browser for a client, you don't need Microsoft on the client, and if Microsoft doesn't control the client, they can't push users towards their server-based applications.

I think Microsoft will have a hard time keeping the genie in the bottle. There will be too many different types of clients for them to control them all. And if Microsoft's applications only work with some clients, competitors will be able to trump them by offering applications that work from any client. [14]

In a world of Web-based applications, there is no automatic place for Microsoft. They may succeed in making themselves a place, but I don't think they'll dominate this new world as they did the world of desktop applications.

It's not so much that a competitor will trip them up as that they will trip over themselves. With the rise of Web-based software, they will be facing not just technical problems but their own wishful thinking. What they need to do is cannibalize their existing business, and I can't see them facing that. The same single-mindedness that has brought them this far will now be working against them. IBM was in exactly the same situation, and they could not master it. IBM made a late and half-hearted entry into the microcomputer business because they were ambivalent about threatening their cash cow, mainframe computing. Microsoft will likewise be hampered by wanting to save the desktop. A cash cow can be a damned heavy monkey on your back.

I'm not saying that no one will dominate server-based applications. Someone probably will eventually. But I think that there will be a good long period of cheerful chaos, just as there was in the early days of microcomputers. That was a good time for startups. Lots of small companies flourished, and did it by making cool things.

Startups but More So

The classic startup is fast and informal, with few people and little money. Those few people work very hard, and technology magnifies the effect of the decisions they make. If they win, they win big.

In a startup writing Web-based applications, everything you associate with startups is taken to an extreme. You can write and launch a product with even fewer people and even less money. You have to be even faster, and you can get away with being more informal. You can literally launch your product as three guys sitting in the living room of an apartment, and a server collocated at an ISP. We did.

Over time the teams have gotten smaller, faster, and more informal. In 1960, software development meant a roomful of men with horn rimmed glasses and narrow black neckties, industriously writing ten lines of code a day on IBM coding forms. In 1980, it was a team of eight to ten people wearing jeans to the office and typing into vt100s. Now it's a couple of guys sitting in a living room with laptops. (And jeans turn out not to be the last word in informality.)

Startups are stressful, and this, unfortunately, is also taken to an extreme with Web-based applications. Many software companies, especially at the beginning, have periods where the developers slept under their desks and so on. The alarming thing about Web-based software is that there is nothing to prevent this becoming the default. The stories about sleeping under desks usually end: then at last we shipped it and we all went home and slept for a week. Web-based software never ships. You can work 16-hour days for as long as you want to. And because you can, and your competitors can, you tend to be forced to. You can, so you must. It's Parkinson's Law running in reverse.

The worst thing is not the hours but the responsibility. Programmers and system administrators traditionally each have their own separate worries. Programmers have to worry about bugs, and system administrators have to worry about infrastructure. Programmers may spend a long day up to their elbows in source code, but at some point they get to go home and forget about it. System administrators never quite leave the job behind, but when they do get paged at 4:00 AM, they don't usually have to do anything very complicated. With Web-based applications, these two kinds of stress get combined. The programmers become system administrators, but without the sharply defined limits that ordinarily make the job bearable.

At Viaweb we spent the first six months just writing software. We worked the usual long hours of an early startup. In a desktop software company, this would have been the part where we were working hard, but it felt like a vacation compared to the next phase, when we took users onto our server. The second biggest benefit of selling Viaweb to Yahoo (after the money) was to be able to dump ultimate responsibility for the whole thing onto the shoulders of a big company.

Desktop software forces users to become system administrators. Web-based software forces programmers to. There is less stress in total, but more for the programmers. That's not necessarily bad news. If you're a startup competing with a big company, it's good news. [15] Web-based applications offer a straightforward way to outwork your competitors. No startup asks for more.

Just Good Enough

One thing that might deter you from writing Web-based applications is the lameness of Web pages as a UI. That is a problem, I admit. There were a few things we would have really liked to add to HTML and HTTP. What matters, though, is that Web pages are just good enough.

There is a parallel here with the first microcomputers. The processors in those machines weren't actually intended to be the CPUs of computers. They were designed to be used in things like traffic lights. But guys like Ed Roberts, who designed the Altair, realized that they were just good enough. You could combine one of these chips with some memory (256 bytes in the first Altair), and front panel switches, and you'd have a working computer. Being able to have your own computer was so exciting that there were plenty of people who wanted to buy them, however limited.

Web pages weren't designed to be a UI for applications, but they're just good enough. And for a significant number of users, software that you can use from any browser will be enough of a win in itself to outweigh any awkwardness in the UI. Maybe you can't write the best-looking spreadsheet using HTML, but you can write a spreadsheet that

several people can use simultaneously from different locations without special client software, or that can incorporate live data feeds, or that can page you when certain conditions are triggered. More importantly, you can write new kinds of applications that don't even have names yet. VisiCalc was not merely a microcomputer version of a mainframe application, after all-- it was a new type of application.

Of course, server-based applications don't have to be Web-based. You could have some other kind of client. But I'm pretty sure that's a bad idea. It would be very convenient if you could assume that everyone would install your client-- so convenient that you could easily convince yourself that they all would-- but if they don't, you're hosed. Because Web-based software assumes nothing about the client, it will work anywhere the Web works. That's a big advantage already, and the advantage will grow as new Web devices proliferate. Users will like you because your software just works, and your life will be easier because you won't have to tweak it for every new client. [16]

I feel like I've watched the evolution of the Web as closely as anyone, and I can't predict what's going to happen with clients. Convergence is probably coming, but where? I can't pick a winner. One thing I can predict is conflict between AOL and Microsoft. Whatever Microsoft's .NET turns out to be, it will probably involve connecting the desktop to servers. Unless AOL fights back, they will either be pushed aside or turned into a pipe between Microsoft client and server software. If Microsoft and AOL get into a client war, the only thing sure to work on both will be browsing the Web, meaning Web-based applications will be the only kind that work everywhere.

How will it all play out? I don't know. And you don't have to know if you bet on Web-based applications. No one can break that without breaking browsing. The Web may not be the only way to deliver software, but it's one that works now and will continue to work for a long time. Web-based applications are cheap to develop, and easy for even the smallest startup to deliver. They're a lot of work, and of a particularly stressful kind, but that only makes the odds better for startups.

Why Not?

E. B. White was amused to learn from a farmer friend that many electrified fences don't have any current running through them. The cows apparently learn to stay away from them, and after that you don't need the current. "Rise up, cows!" he wrote, "Take your liberty while despots snore!"

If you're a hacker who has thought of one day starting a startup, there are probably two things keeping you from doing it. One is that you don't know anything about business. The other is that you're afraid of competition. Neither of these fences have any current in them.

There are only two things you have to know about business: build something users love, and make more than you spend. If you get these two right, you'll be ahead of most startups. You can figure out the rest as you go.

You may not at first make more than you spend, but as long as the gap is closing fast enough you'll be ok. If you start out underfunded, it will at least encourage a habit of frugality. The less you spend, the easier it is to make more than you spend. Fortunately, it can be very cheap to launch a Web-based application. We launched on under $10,000, and

it would be even cheaper today. We had to spend thousands on a server, and thousands more to get SSL. (The only company selling SSL software at the time was Netscape.) Now you can rent a much more powerful server, with SSL included, for less than we paid for bandwidth alone. You could launch a Web-based application now for less than the cost of a fancy office chair.

As for building something users love, here are some general tips. Start by making something clean and simple that you would want to use yourself. Get a version 1.0 out fast, then continue to improve the software, listening closely to the users as you do. The customer is always right, but different customers are right about different things; the least sophisticated users show you what you need to simplify and clarify, and the most sophisticated tell you what features you need to add. The best thing software can be is easy, but the way to do this is to get the defaults right, not to limit users' choices. Don't get complacent if your competitors' software is lame; the standard to compare your software to is what it could be, not what your current competitors happen to have. Use your software yourself, all the time. Viaweb was supposed to be an online store builder, but we used it to make our own site too. Don't listen to marketing people or designers or product managers just because of their job titles. If they have good ideas, use them, but it's up to you to decide; software has to be designed by hackers who understand design, not designers who know a little about software. If you can't design software as well as implement it, don't start a startup.

Now let's talk about competition. What you're afraid of is not presumably groups of hackers like you, but actual companies, with offices and business plans and salesmen and so on, right? Well, they are more afraid of you than you are of them, and they're right. It's a lot easier for a couple of hackers to figure out how to rent office space or hire sales people than it is for a company of any size to get software written. I've been on both sides, and I know. When Viaweb was bought by Yahoo, I suddenly found myself working for a big company, and it was like trying to run through waist-deep water.

I don't mean to disparage Yahoo. They had some good hackers, and the top management were real butt-kickers. For a big company, they were exceptional. But they were still only about a tenth as productive as a small startup. No big company can do much better than that. What's scary about Microsoft is that a company so big can develop software at all. They're like a mountain that can walk.

Don't be intimidated. You can do as much that Microsoft can't as they can do that you can't. And no one can stop you. You don't have to ask anyone's permission to develop Web-based applications. You don't have to do licensing deals, or get shelf space in retail stores, or grovel to have your application bundled with the OS. You can deliver software right to the browser, and no one can get between you and potential users without preventing them from browsing the Web.

You may not believe it, but I promise you, Microsoft is scared of you. The complacent middle managers may not be, but Bill is, because he was you once, back in 1975, the last time a new way of delivering software appeared.

Notes

[1] Realizing that much of the money is in the services, companies building lightweight clients have usually tried to combine the hardware with an online service. This approach has not worked well, partly because you need two different kinds of companies to build consumer electronics and to run an online service, and partly because users hate the idea. Giving away the razor and making money on the blades may work for Gillette, but a razor is much smaller commitment than a Web terminal. Cell phone handset makers are satisfied to sell hardware without trying to capture the service revenue as well. That should probably be the model for Internet clients too. If someone just sold a nice-looking little box with a Web browser that you could use to connect through any ISP, every technophobe in the country would buy one.

[2] Security always depends more on not screwing up than any design decision, but the nature of server-based software will make developers pay more attention to not screwing up. Compromising a server could cause such damage that ASPs (that want to stay in business) are likely to be careful about security.

[3] In 1995, when we started Viaweb, Java applets were supposed to be the technology everyone was going to use to develop server-based applications. Applets seemed to us an old-fashioned idea. Download programs to run on the client? Simpler just to go all the way and run the programs on the server. We wasted little time on applets, but countless other startups must have been lured into this tar pit. Few can have escaped alive, or Microsoft could not have gotten away with dropping Java in the most recent version of Explorer.

[4] This point is due to Trevor Blackwell, who adds "the cost of writing software goes up more than linearly with its size. Perhaps this is mainly due to fixing old bugs, and the cost can be more linear if all bugs are found quickly."

[5] The hardest kind of bug to find may be a variant of compound bug where one bug happens to compensate for another. When you fix one bug, the other becomes visible. But it will seem as if the fix is at fault, since that was the last thing you changed.

[6] Within Viaweb we once had a contest to describe the worst thing about our software. Two customer support people tied for first prize with entries I still shiver to recall. We fixed both problems immediately.

[7] Robert Morris wrote the ordering system, which shoppers used to place orders. Trevor Blackwell wrote the image generator and the manager, which merchants used to retrieve orders, view statistics, and configure domain names etc. I wrote the editor, which merchants used to build their sites. The ordering system and image generator were written in C and C++, the manager mostly in Perl, and the editor in Lisp.

[8] Price discrimination is so pervasive (how often have you heard a retailer claim that their buying power meant lower prices for you?) that I was surprised to find it was outlawed in the U.S. by the Robinson-Patman Act of 1936. This law does not appear to be vigorously enforced.

[9] In No Logo, Naomi Klein says that clothing brands favored by "urban youth" do not try too hard to prevent shoplifting because in their target market the shoplifters are also the fashion leaders.

[10] Companies often wonder what to outsource and what not to. One possible answer: outsource any job that's not directly exposed to competitive pressure, because outsourcing it will thereby expose it to competitive pressure.

[11] The two guys were Dan Bricklin and Bob Frankston. Dan wrote a prototype in Basic in a couple days, then over the course of the next year they worked together (mostly at night) to make a more powerful version written in 6502 machine language. Dan was at Harvard Business School at the time and Bob nominally had a day job writing software. "There was no great risk in doing a business," Bob wrote, "If it failed it failed. No big deal."

[12] It's not quite as easy as I make it sound. It took a painfully long time for word of mouth to get going, and we did not start to get a lot of press coverage until we hired a PR firm (admittedly the best in the business) for $16,000 per month. However, it was true that the only significant channel was our own Web site.

[13] If the Mac was so great, why did it lose? Cost, again. Microsoft concentrated on the software business, and unleashed a swarm of cheap component suppliers on Apple hardware. It did not help, either, that suits took over during a critical period.

[14] One thing that would help Web-based applications, and help keep the next generation of software from being overshadowed by Microsoft, would be a good open-source browser. Mozilla is open-source but seems to have suffered from having been corporate software for so long. A small, fast browser that was actively maintained would be a great thing in itself, and would probably also encourage companies to build little Web appliances.

Among other things, a proper open-source browser would cause HTTP and HTML to continue to evolve (as e.g. Perl has). It would help Web-based applications greatly to be able to distinguish between selecting a link and following it; all you'd need to do this would be a trivial enhancement of HTTP, to allow multiple urls in a request. Cascading menus would also be good.

If you want to change the world, write a new Mosaic. Think it's too late? In 1998 a lot of people thought it was too late to launch a new search engine, but Google proved them wrong. There is always room for something new if the current options suck enough. Make sure it works on all the free OSes first-- new things start with their users.

[15] Trevor Blackwell, who probably knows more about this from personal experience than anyone, writes:

"I would go farther in saying that because server-based software is so hard on the programmers, it causes a fundamental economic shift away from large companies. It requires the kind of intensity and dedication from programmers that they will only be willing to provide when it's their own company. Software companies can hire

skilled people to work in a not-too-demanding environment, and can hire unskilled people to endure hardships, but they can't hire highly skilled people to bust their asses. Since capital is no longer needed, big companies have little to bring to the table."

[16] In the original version of this essay, I advised avoiding Javascript. That was a good plan in 2001, but Javascript now works.

How Not to Die

Want to start a startup? Get funded by Y Combinator.

August 2007

(This is a talk I gave at the last Y Combinator dinner of the summer. Usually we don't have a speaker at the last dinner; it's more of a party. But it seemed worth spoiling the atmosphere if I could save some of the startups from preventable deaths. So at the last minute I cooked up this rather grim talk. I didn't mean this as an essay; I wrote it down because I only had two hours before dinner and think fastest while writing.)

A couple days ago I told a reporter that we expected about a third of the companies we funded to succeed. Actually I was being conservative. I'm hoping it might be as much as a half. Wouldn't it be amazing if we could achieve a 50% success rate?

Another way of saying that is that half of you are going to die. Phrased that way, it doesn't sound good at all. In fact, it's kind of weird when you think about it, because our definition of success is that the founders get rich. If half the startups we fund succeed, then half of you are going to get rich and the other half are going to get nothing.

If you can just avoid dying, you get rich. That sounds like a joke, but it's actually a pretty good description of what happens in a typical startup. It certainly describes what happened in Viaweb. We avoided dying till we got rich.

It was really close, too. When we were visiting Yahoo to talk about being acquired, we had to interrupt everything and borrow one of their conference rooms to talk down an investor who was about to back out of a new funding round we needed to stay alive. So even in the middle of getting rich we were fighting off the grim reaper.

You may have heard that quote about luck consisting of opportunity meeting preparation. You've now done the preparation. The work you've done so far has, in effect, put you in a position to get lucky: you can now get rich by not letting your company die. That's more than most people have. So let's talk about how not to die.

We've done this five times now, and we've seen a bunch of startups die. About 10 of them so far. We don't know exactly what happens when they die, because they generally don't die loudly and heroically. Mostly they crawl off somewhere and die.

For us the main indication of impending doom is when we don't hear from you. When we haven't heard from, or about, a startup for a couple months, that's a bad sign. If we send them an email asking what's up, and they don't reply, that's a really bad sign. So far that is a 100% accurate predictor of death.

Whereas if a startup regularly does new deals and releases and either sends us mail or shows up at YC events, they're probably going to live.

I realize this will sound naive, but maybe the linkage works in both directions. Maybe if you can arrange that we keep hearing from you, you won't die.

That may not be so naive as it sounds. You've probably noticed that having dinners every Tuesday with us and the other founders causes you to get more done than you would otherwise, because every dinner is a mini Demo Day. Every dinner is a kind of a deadline. So the mere constraint of staying in regular contact with us will push you to make things happen, because otherwise you'll be embarrassed to tell us that you haven't done anything new since the last time we talked.

If this works, it would be an amazing hack. It would be pretty cool if merely by staying in regular contact with us you could get rich. It sounds crazy, but there's a good chance that would work.

A variant is to stay in touch with other YC-funded startups. There is now a whole neighborhood of them in San Francisco. If you move there, the peer pressure that made you work harder all summer will continue to operate.

When startups die, the official cause of death is always either running out of money or a critical founder bailing. Often the two occur simultaneously. But I think the underlying cause is usually that they've become demoralized. You rarely hear of a startup that's working around the clock doing deals and pumping out new features, and dies because they can't pay their bills and their ISP unplugs their server.

Startups rarely die in mid keystroke. So keep typing!

If so many startups get demoralized and fail when merely by hanging on they could get rich, you have to assume that running a startup can be demoralizing. That is certainly true. I've been there, and that's why I've never done another startup. The low points in a startup are just unbelievably low. I bet even Google had moments where things seemed hopeless.

Knowing that should help. If you know it's going to feel terrible sometimes, then when it feels terrible you won't think "ouch, this feels terrible, I give up." It feels that way for everyone. And if you just hang on, things will probably get better. The metaphor people use to describe the way a startup feels is at least a roller coaster and not drowning. You don't just sink and sink; there are ups after the downs.

Another feeling that seems alarming but is in fact normal in a startup is the feeling that what you're doing isn't working. The reason you can expect to feel this is that what you do probably won't work. Startups almost never get it right the first time. Much more commonly you launch something, and no one cares. Don't assume when this happens that you've failed. That's normal for startups. But don't sit around doing nothing. Iterate.

I like Paul Buchheit's suggestion of trying to make something that at least someone really loves. As long as you've made something that a few users are ecstatic about, you're on the right track. It will be good for your morale to have even a handful of users who really love you, and startups run on morale. But also it will tell you what to focus on. What is it about you that they love? Can you do more of that? Where can you find more people who love that sort of thing? As long as you have some core of users who love you, all you have to do is expand it. It may take a while, but as long as you keep plugging away, you'll win in the end. Both Blogger and Delicious did that. Both took years to succeed. But both began with a core of fanatically devoted users, and all Evan and Joshua had to do was grow that core incrementally. Wufoo is on the same trajectory now.

So when you release something and it seems like no one cares, look more closely. Are there zero users who really love you, or is there at least some little group that does? It's quite possible there will be zero. In that case, tweak your product and try again. Every one of you is working on a space that contains at least one winning permutation somewhere in it. If you just keep trying, you'll find it.

Let me mention some things not to do. The number one thing not to do is other things. If you find yourself saying a sentence that ends with "but we're going to keep working on the startup," you are in big trouble. Bob's going to grad school, but we're going to keep working on the startup. We're moving back to Minnesota, but we're going to keep working on the startup. We're taking on some consulting projects, but we're going to keep working on the startup. You may as well just translate these to "we're giving up on the startup, but we're not willing to admit that to ourselves," because that's what it means most of the time. A startup is so hard that working on it can't be preceded by "but."

In particular, don't go to graduate school, and don't start other projects. Distraction is fatal to startups. Going to (or back to) school is a huge predictor of death because in addition to the distraction it gives you something to say you're doing. If you're only doing a startup, then if the startup fails, you fail. If you're in grad school and your startup fails, you can say later "Oh yeah, we had this startup on the side when I was in grad school, but it didn't go anywhere."

You can't use euphemisms like "didn't go anywhere" for something that's your only occupation. People won't let you.

One of the most interesting things we've discovered from working on Y Combinator is that founders are more motivated by the fear of looking bad than by the hope of getting millions of dollars. So if you want to get millions of dollars, put yourself in a position where failure will be public and humiliating.

When we first met the founders of Octopart, they seemed very smart, but not a great bet to succeed, because they didn't seem especially committed. One of the two founders was still in grad school. It was the usual story: he'd drop out if it looked like the startup was taking off. Since then he has not only dropped out of grad school, but appeared

full length in Newsweek with the word "Billionaire" printed across his chest. He just cannot fail now. Everyone he knows has seen that picture. Girls who dissed him in high school have seen it. His mom probably has it on the fridge. It would be unthinkably humiliating to fail now. At this point he is committed to fight to the death.

I wish every startup we funded could appear in a Newsweek article describing them as the next generation of billionaires, because then none of them would be able to give up. The success rate would be 90%. I'm not kidding.

When we first knew the Octoparts they were lighthearted, cheery guys. Now when we talk to them they seem grimly determined. The electronic parts distributors are trying to squash them to keep their monopoly pricing. (If it strikes you as odd that people still order electronic parts out of thick paper catalogs in 2007, there's a reason for that. The distributors want to prevent the transparency that comes from having prices online.) I feel kind of bad that we've transformed these guys from lighthearted to grimly determined. But that comes with the territory. If a startup succeeds, you get millions of dollars, and you don't get that kind of money just by asking for it. You have to assume it takes some amount of pain.

And however tough things get for the Octoparts, I predict they'll succeed. They may have to morph themselves into something totally different, but they won't just crawl off and die. They're smart; they're working in a promising field; and they just cannot give up.

All of you guys already have the first two. You're all smart and working on promising ideas. Whether you end up among the living or the dead comes down to the third ingredient, not giving up.

So I'll tell you now: bad shit is coming. It always is in a startup. The odds of getting from launch to liquidity without some kind of disaster happening are one in a thousand. So don't get demoralized. When the disaster strikes, just say to yourself, ok, this was what Paul was talking about. What did he say to do? Oh, yeah. Don't give up.

What Business Can Learn from Open Source

August 2005

(This essay is derived from a talk at Oscon 2005.)

Lately companies have been paying more attention to open source. Ten years ago there seemed a real danger Microsoft would extend its monopoly to servers. It seems safe to say now that open source has prevented that. A recent survey found 52% of companies are replacing Windows servers with Linux servers. [1]

More significant, I think, is which 52% they are. At this point, anyone proposing to run Windows on servers should be prepared to explain what they know about servers that Google, Yahoo, and Amazon don't.

But the biggest thing business has to learn from open source is not about Linux or Firefox, but about the forces that produced them. Ultimately these will affect a lot more than what software you use.

We may be able to get a fix on these underlying forces by triangulating from open source and blogging. As you've probably noticed, they have a lot in common.

Like open source, blogging is something people do themselves, for free, because they enjoy it. Like open source hackers, bloggers compete with people working for money, and often win. The method of ensuring quality is also the same: Darwinian. Companies ensure quality through rules to prevent employees from screwing up. But you don't need that when the audience can communicate with one another. People just produce whatever they want; the good stuff spreads, and the bad gets ignored. And in both cases, feedback from the audience improves the best work.

Another thing blogging and open source have in common is the Web. People have always been willing to do great work for free, but before the Web it was harder to reach an audience or collaborate on projects.

Amateurs

I think the most important of the new principles business has to learn is that people work a lot harder on stuff they like. Well, that's news to no one. So how can I claim business has to learn it? When I say business doesn't know this, I mean the structure of business doesn't reflect it.

Business still reflects an older model, exemplified by the French word for working: travailler. It has an English cousin, travail, and what it means is torture. [2]

This turns out not to be the last word on work, however. As societies get richer, they learn something about work that's a lot like what they learn about diet. We know now that the healthiest diet is the one our peasant ancestors were forced to eat because they were poor. Like rich food, idleness only seems desirable when you don't get enough of it. I think we were designed to work, just as we were designed to eat a certain amount of fiber, and we feel bad if we don't.

There's a name for people who work for the love of it: amateurs. The word now has such bad connotations that we forget its etymology, though it's staring us in the face. "Amateur" was originally rather a complimentary word. But the thing to be in the twentieth century was professional, which amateurs, by definition, are not.

That's why the business world was so surprised by one lesson from open source: that people working for love often surpass those working for money. Users don't switch from Explorer to Firefox because they want to hack the source. They switch because it's a better browser.

It's not that Microsoft isn't trying. They know controlling the browser is one of the keys to retaining their monopoly. The problem is the same they face in operating systems: they can't pay people enough to build something better than a group of inspired hackers will build for free.

I suspect professionalism was always overrated-- not just in the literal sense of working for money, but also connotations like formality and detachment. Inconceivable as it would have seemed in, say, 1970, I think professionalism was largely a fashion, driven by conditions that happened to exist in the twentieth century.

One of the most powerful of those was the existence of "channels." Revealingly, the same term was used for both products and information: there were distribution channels, and TV and radio channels.

It was the narrowness of such channels that made professionals seem so superior to amateurs. There were only a few jobs as professional journalists, for example, so competition ensured the average journalist was fairly good. Whereas anyone can express opinions about current events in a bar. And so the average person expressing his opinions in a bar sounds like an idiot compared to a journalist writing about the subject.

On the Web, the barrier for publishing your ideas is even lower. You don't have to buy a drink, and they even let kids in. Millions of people are publishing online, and the average level of what they're writing, as you might expect, is not very good. This has led some in the media to conclude that blogs don't present much of a threat-- that blogs are just a fad.

Actually, the fad is the word "blog," at least the way the print media now use it. What they mean by "blogger" is not someone who publishes in a weblog format, but anyone who publishes online. That's going to become a problem as

the Web becomes the default medium for publication. So I'd like to suggest an alternative word for someone who publishes online. How about "writer?"

Those in the print media who dismiss the writing online because of its low average quality are missing an important point: no one reads the average blog. In the old world of channels, it meant something to talk about average quality, because that's what you were getting whether you liked it or not. But now you can read any writer you want. So the average quality of writing online isn't what the print media are competing against. They're competing against the best writing online. And, like Microsoft, they're losing.

I know that from my own experience as a reader. Though most print publications are online, I probably read two or three articles on individual people's sites for every one I read on the site of a newspaper or magazine.

And when I read, say, New York Times stories, I never reach them through the Times front page. Most I find through aggregators like Google News or Slashdot or Delicious. Aggregators show how much better you can do than the channel. The New York Times front page is a list of articles written by people who work for the New York Times. Delicious is a list of articles that are interesting. And it's only now that you can see the two side by side that you notice how little overlap there is.

Most articles in the print media are boring. For example, the president notices that a majority of voters now think invading Iraq was a mistake, so he makes an address to the nation to drum up support. Where is the man bites dog in that? I didn't hear the speech, but I could probably tell you exactly what he said. A speech like that is, in the most literal sense, not news: there is nothing new in it. [3]

Nor is there anything new, except the names and places, in most "news" about things going wrong. A child is abducted; there's a tornado; a ferry sinks; someone gets bitten by a shark; a small plane crashes. And what do you learn about the world from these stories? Absolutely nothing. They're outlying data points; what makes them gripping also makes them irrelevant.

As in software, when professionals produce such crap, it's not surprising if amateurs can do better. Live by the channel, die by the channel: if you depend on an oligopoly, you sink into bad habits that are hard to overcome when you suddenly get competition. [4]

Workplaces

Another thing blogs and open source software have in common is that they're often made by people working at home. That may not seem surprising. But it should be. It's the architectural equivalent of a home-made aircraft shooting down an F-18. Companies spend millions to build office buildings for a single purpose: to be a place to work. And yet people working in their own homes, which aren't even designed to be workplaces, end up being more productive.

This proves something a lot of us have suspected. The average office is a miserable place to get work done. And a lot of what makes offices bad are the very qualities we associate with professionalism. The sterility of offices is supposed to suggest efficiency. But suggesting efficiency is a different thing from actually being efficient.

The atmosphere of the average workplace is to productivity what flames painted on the side of a car are to speed. And it's not just the way offices look that's bleak. The way people act is just as bad.

Things are different in a startup. Often as not a startup begins in an apartment. Instead of matching beige cubicles they have an assortment of furniture they bought used. They work odd hours, wearing the most casual of clothing. They look at whatever they want online without worrying whether it's "work safe." The cheery, bland language of the office is replaced by wicked humor. And you know what? The company at this stage is probably the most productive it's ever going to be.

Maybe it's not a coincidence. Maybe some aspects of professionalism are actually a net lose.

To me the most demoralizing aspect of the traditional office is that you're supposed to be there at certain times. There are usually a few people in a company who really have to, but the reason most employees work fixed hours is that the company can't measure their productivity.

The basic idea behind office hours is that if you can't make people work, you can at least prevent them from having fun. If employees have to be in the building a certain number of hours a day, and are forbidden to do non-work things while there, then they must be working. In theory. In practice they spend a lot of their time in a no-man's land, where they're neither working nor having fun.

If you could measure how much work people did, many companies wouldn't need any fixed workday. You could just say: this is what you have to do. Do it whenever you like, wherever you like. If your work requires you to talk to other people in the company, then you may need to be here a certain amount. Otherwise we don't care.

That may seem utopian, but it's what we told people who came to work for our company. There were no fixed office hours. I never showed up before 11 in the morning. But we weren't saying this to be benevolent. We were saying: if you work here we expect you to get a lot done. Don't try to fool us just by being here a lot.

The problem with the facetime model is not just that it's demoralizing, but that the people pretending to work interrupt the ones actually working. I'm convinced the facetime model is the main reason large organizations have so many meetings. Per capita, large organizations accomplish very little. And yet all those people have to be on site at least eight hours a day. When so much time goes in one end and so little achievement comes out the other, something has to give. And meetings are the main mechanism for taking up the slack.

For one year I worked at a regular nine to five job, and I remember well the strange, cozy feeling that comes over one during meetings. I was very aware, because of the novelty, that I was being paid for programming. It seemed just

amazing, as if there was a machine on my desk that spat out a dollar bill every two minutes no matter what I did. Even while I was in the bathroom! But because the imaginary machine was always running, I felt I always ought to be working. And so meetings felt wonderfully relaxing. They counted as work, just like programming, but they were so much easier. All you had to do was sit and look attentive.

Meetings are like an opiate with a network effect. So is email, on a smaller scale. And in addition to the direct cost in time, there's the cost in fragmentation-- breaking people's day up into bits too small to be useful.

You can see how dependent you've become on something by removing it suddenly. So for big companies I propose the following experiment. Set aside one day where meetings are forbidden-- where everyone has to sit at their desk all day and work without interruption on things they can do without talking to anyone else. Some amount of communication is necessary in most jobs, but I'm sure many employees could find eight hours worth of stuff they could do by themselves. You could call it "Work Day."

The other problem with pretend work is that it often looks better than real work. When I'm writing or hacking I spend as much time just thinking as I do actually typing. Half the time I'm sitting drinking a cup of tea, or walking around the neighborhood. This is a critical phase-- this is where ideas come from-- and yet I'd feel guilty doing this in most offices, with everyone else looking busy.

It's hard to see how bad some practice is till you have something to compare it to. And that's one reason open source, and even blogging in some cases, are so important. They show us what real work looks like.

We're funding eight new startups at the moment. A friend asked what they were doing for office space, and seemed surprised when I said we expected them to work out of whatever apartments they found to live in. But we didn't propose that to save money. We did it because we want their software to be good. Working in crappy informal spaces is one of the things startups do right without realizing it. As soon as you get into an office, work and life start to drift apart.

That is one of the key tenets of professionalism. Work and life are supposed to be separate. But that part, I'm convinced, is a mistake.

Bottom-Up

The third big lesson we can learn from open source and blogging is that ideas can bubble up from the bottom, instead of flowing down from the top. Open source and blogging both work bottom-up: people make what they want, and the best stuff prevails.

Does this sound familiar? It's the principle of a market economy. Ironically, though open source and blogs are done for free, those worlds resemble market economies, while most companies, for all their talk about the value of free markets, are run internally like communist states.

There are two forces that together steer design: ideas about what to do next, and the enforcement of quality. In the channel era, both flowed down from the top. For example, newspaper editors assigned stories to reporters, then edited what they wrote.

Open source and blogging show us things don't have to work that way. Ideas and even the enforcement of quality can flow bottom-up. And in both cases the results are not merely acceptable, but better. For example, open source software is more reliable precisely because it's open source; anyone can find mistakes.

The same happens with writing. As we got close to publication, I found I was very worried about the essays in Hackers & Painters that hadn't been online. Once an essay has had a couple thousand page views I feel reasonably confident about it. But these had had literally orders of magnitude less scrutiny. It felt like releasing software without testing it.

That's what all publishing used to be like. If you got ten people to read a manuscript, you were lucky. But I'd become so used to publishing online that the old method now seemed alarmingly unreliable, like navigating by dead reckoning once you'd gotten used to a GPS.

The other thing I like about publishing online is that you can write what you want and publish when you want. Earlier this year I wrote something that seemed suitable for a magazine, so I sent it to an editor I know. As I was waiting to hear back, I found to my surprise that I was hoping they'd reject it. Then I could put it online right away. If they accepted it, it wouldn't be read by anyone for months, and in the meantime I'd have to fight word-by-word to save it from being mangled by some twenty five year old copy editor. [5]

Many employees would like to build great things for the companies they work for, but more often than not management won't let them. How many of us have heard stories of employees going to management and saying, please let us build this thing to make money for you-- and the company saying no? The most famous example is probably Steve Wozniak, who originally wanted to build microcomputers for his then-employer, HP. And they turned him down. On the blunderometer, this episode ranks with IBM accepting a non-exclusive license for DOS. But I think this happens all the time. We just don't hear about it usually, because to prove yourself right you have to quit and start your own company, like Wozniak did.

Startups

So these, I think, are the three big lessons open source and blogging have to teach business: (1) that people work harder on stuff they like, (2) that the standard office environment is very unproductive, and (3) that bottom-up often works better than top-down.

I can imagine managers at this point saying: what is this guy talking about? What good does it do me to know that my programmers would be more productive working at home on their own projects? I need their asses in here working on version 3.2 of our software, or we're never going to make the release date.

And it's true, the benefit that specific manager could derive from the forces I've described is near zero. When I say business can learn from open source, I don't mean any specific business can. I mean business can learn about new conditions the same way a gene pool does. I'm not claiming companies can get smarter, just that dumb ones will die.

So what will business look like when it has assimilated the lessons of open source and blogging? I think the big obstacle preventing us from seeing the future of business is the assumption that people working for you have to be employees. But think about what's going on underneath: the company has some money, and they pay it to the employee in the hope that he'll make something worth more than they paid him. Well, there are other ways to arrange that relationship. Instead of paying the guy money as a salary, why not give it to him as investment? Then instead of coming to your office to work on your projects, he can work wherever he wants on projects of his own.

Because few of us know any alternative, we have no idea how much better we could do than the traditional employer-employee relationship. Such customs evolve with glacial slowness. Our employer-employee relationship still retains a big chunk of master-servant DNA. [6]

I dislike being on either end of it. I'll work my ass off for a customer, but I resent being told what to do by a boss. And being a boss is also horribly frustrating; half the time it's easier just to do stuff yourself than to get someone else to do it for you. I'd rather do almost anything than give or receive a performance review.

On top of its unpromising origins, employment has accumulated a lot of cruft over the years. The list of what you can't ask in job interviews is now so long that for convenience I assume it's infinite. Within the office you now have to walk on eggshells lest anyone say or do something that makes the company prey to a lawsuit. And God help you if you fire anyone.

Nothing shows more clearly that employment is not an ordinary economic relationship than companies being sued for firing people. In any purely economic relationship you're free to do what you want. If you want to stop buying steel pipe from one supplier and start buying it from another, you don't have to explain why. No one can accuse you of unjustly switching pipe suppliers. Justice implies some kind of paternal obligation that isn't there in transactions between equals.

Most of the legal restrictions on employers are intended to protect employees. But you can't have action without an equal and opposite reaction. You can't expect employers to have some kind of paternal responsibility toward employees without putting employees in the position of children. And that seems a bad road to go down.

Next time you're in a moderately large city, drop by the main post office and watch the body language of the people working there. They have the same sullen resentment as children made to do something they don't want to. Their union has exacted pay increases and work restrictions that would have been the envy of previous generations of

postal workers, and yet they don't seem any happier for it. It's demoralizing to be on the receiving end of a paternalistic relationship, no matter how cozy the terms. Just ask any teenager.

I see the disadvantages of the employer-employee relationship because I've been on both sides of a better one: the investor-founder relationship. I wouldn't claim it's painless. When I was running a startup, the thought of our investors used to keep me up at night. And now that I'm an investor, the thought of our startups keeps me up at night. All the pain of whatever problem you're trying to solve is still there. But the pain hurts less when it isn't mixed with resentment.

I had the misfortune to participate in what amounted to a controlled experiment to prove that. After Yahoo bought our startup I went to work for them. I was doing exactly the same work, except with bosses. And to my horror I started acting like a child. The situation pushed buttons I'd forgotten I had.

The big advantage of investment over employment, as the examples of open source and blogging suggest, is that people working on projects of their own are enormously more productive. And a startup is a project of one's own in two senses, both of them important: it's creatively one's own, and also economically ones's own.

Google is a rare example of a big company in tune with the forces I've described. They've tried hard to make their offices less sterile than the usual cube farm. They give employees who do great work large grants of stock to simulate the rewards of a startup. They even let hackers spend 20% of their time on their own projects.

Why not let people spend 100% of their time on their own projects, and instead of trying to approximate the value of what they create, give them the actual market value? Impossible? That is in fact what venture capitalists do.

So am I claiming that no one is going to be an employee anymore-- that everyone should go and start a startup? Of course not. But more people could do it than do it now. At the moment, even the smartest students leave school thinking they have to get a job. Actually what they need to do is make something valuable. A job is one way to do that, but the more ambitious ones will ordinarily be better off taking money from an investor than an employer.

Hackers tend to think business is for MBAs. But business administration is not what you're doing in a startup. What you're doing is business creation. And the first phase of that is mostly product creation-- that is, hacking. That's the hard part. It's a lot harder to create something people love than to take something people love and figure out how to make money from it.

Another thing that keeps people away from starting startups is the risk. Someone with kids and a mortgage should think twice before doing it. But most young hackers have neither.

And as the example of open source and blogging suggests, you'll enjoy it more, even if you fail. You'll be working on your own thing, instead of going to some office and doing what you're told. There may be more pain in your own company, but it won't hurt as much.

That may be the greatest effect, in the long run, of the forces underlying open source and blogging: finally ditching the old paternalistic employer-employee relationship, and replacing it with a purely economic one, between equals.

Notes

[1] Survey by Forrester Research reported in the cover story of Business Week, 31 Jan 2005. Apparently someone believed you have to replace the actual server in order to switch the operating system.

[2] It derives from the late Latin tripalium, a torture device so called because it consisted of three stakes. I don't know how the stakes were used. "Travel" has the same root.

[3] It would be much bigger news, in that sense, if the president faced unscripted questions by giving a press conference.

[4] One measure of the incompetence of newspapers is that so many still make you register to read stories. I have yet to find a blog that tried that.

[5] They accepted the article, but I took so long to send them the final version that by the time I did the section of the magazine they'd accepted it for had disappeared in a reorganization.

[6] The word "boss" is derived from the Dutch baas, meaning "master."

What the Bubble Got Right

September 2004

(This essay is derived from an invited talk at ICFP 2004.)

I had a front row seat for the Internet Bubble, because I worked at Yahoo during 1998 and 1999. One day, when the stock was trading around $200, I sat down and calculated what I thought the price should be. The answer I got was $12. I went to the next cubicle and told my friend Trevor. "Twelve!" he said. He tried to sound indignant, but he didn't quite manage it. He knew as well as I did that our valuation was crazy.

Yahoo was a special case. It was not just our price to earnings ratio that was bogus. Half our earnings were too. Not in the Enron way, of course. The finance guys seemed scrupulous about reporting earnings. What made our earnings bogus was that Yahoo was, in effect, the center of a Ponzi scheme. Investors looked at Yahoo's earnings and said to themselves, here is proof that Internet companies can make money. So they invested in new startups that promised to be the next Yahoo. And as soon as these startups got the money, what did they do with it? Buy millions of dollars worth of advertising on Yahoo to promote their brand. Result: a capital investment in a startup this quarter shows up as Yahoo earnings next quarter—stimulating another round of investments in startups.

As in a Ponzi scheme, what seemed to be the returns of this system were simply the latest round of investments in it. What made it not a Ponzi scheme was that it was unintentional. At least, I think it was. The venture capital business is pretty incestuous, and there were presumably people in a position, if not to create this situation, to realize what was happening and to milk it.

A year later the game was up. Starting in January 2000, Yahoo's stock price began to crash, ultimately losing 95% of its value.

Notice, though, that even with all the fat trimmed off its market cap, Yahoo was still worth a lot. Even at the morning-after valuations of March and April 2001, the people at Yahoo had managed to create a company worth about $8 billion in just six years.

The fact is, despite all the nonsense we heard during the Bubble about the "new economy," there was a core of truth. You need that to get a really big bubble: you need to have something solid at the center, so that even smart people are sucked in. (Isaac Newton and Jonathan Swift both lost money in the South Sea Bubble of 1720.)

Now the pendulum has swung the other way. Now anything that became fashionable during the Bubble is ipso facto unfashionable. But that's a mistake—an even bigger mistake than believing what everyone was saying in 1999. Over the long term, what the Bubble got right will be more important than what it got wrong.

1. Retail VC

After the excesses of the Bubble, it's now considered dubious to take companies public before they have earnings. But there is nothing intrinsically wrong with that idea. Taking a company public at an early stage is simply retail VC: instead of going to venture capital firms for the last round of funding, you go to the public markets.

By the end of the Bubble, companies going public with no earnings were being derided as "concept stocks," as if it were inherently stupid to invest in them. But investing in concepts isn't stupid; it's what VCs do, and the best of them are far from stupid.

The stock of a company that doesn't yet have earnings is worth something. It may take a while for the market to learn how to value such companies, just as it had to learn to value common stocks in the early 20th century. But markets are good at solving that kind of problem. I wouldn't be surprised if the market ultimately did a better job than VCs do now.

Going public early will not be the right plan for every company. And it can of course be disruptive—by distracting the management, or by making the early employees suddenly rich. But just as the market will learn how to value startups, startups will learn how to minimize the damage of going public.

2. The Internet

The Internet genuinely is a big deal. That was one reason even smart people were fooled by the Bubble. Obviously it was going to have a huge effect. Enough of an effect to triple the value of Nasdaq companies in two years? No, as it turned out. But it was hard to say for certain at the time. [1]

The same thing happened during the Mississippi and South Sea Bubbles. What drove them was the invention of organized public finance (the South Sea Company, despite its name, was really a competitor of the Bank of England). And that did turn out to be a big deal, in the long run.

Recognizing an important trend turns out to be easier than figuring out how to profit from it. The mistake investors always seem to make is to take the trend too literally. Since the Internet was the big new thing, investors supposed that the more Internettish the company, the better. Hence such parodies as Pets.Com.

In fact most of the money to be made from big trends is made indirectly. It was not the railroads themselves that made the most money during the railroad boom, but the companies on either side, like Carnegie's steelworks, which made the rails, and Standard Oil, which used railroads to get oil to the East Coast, where it could be shipped to Europe.

I think the Internet will have great effects, and that what we've seen so far is nothing compared to what's coming. But most of the winners will only indirectly be Internet companies; for every Google there will be ten JetBlues.

3. Choices

Why will the Internet have great effects? The general argument is that new forms of communication always do. They happen rarely (till industrial times there were just speech, writing, and printing), but when they do, they always cause a big splash.

The specific argument, or one of them, is the Internet gives us more choices. In the "old" economy, the high cost of presenting information to people meant they had only a narrow range of options to choose from. The tiny, expensive pipeline to consumers was tellingly named "the channel." Control the channel and you could feed them what you wanted, on your terms. And it was not just big corporations that depended on this principle. So, in their way, did labor unions, the traditional news media, and the art and literary establishments. Winning depended not on doing good work, but on gaining control of some bottleneck.

There are signs that this is changing. Google has over 82 million unique users a month and annual revenues of about three billion dollars. [2] And yet have you ever seen a Google ad? Something is going on here.

Admittedly, Google is an extreme case. It's very easy for people to switch to a new search engine. It costs little effort and no money to try a new one, and it's easy to see if the results are better. And so Google doesn't have to advertise. In a business like theirs, being the best is enough.

The exciting thing about the Internet is that it's shifting everything in that direction. The hard part, if you want to win by making the best stuff, is the beginning. Eventually everyone will learn by word of mouth that you're the best, but how do you survive to that point? And it is in this crucial stage that the Internet has the most effect. First, the Internet lets anyone find you at almost zero cost. Second, it dramatically speeds up the rate at which reputation spreads by word of mouth. Together these mean that in many fields the rule will be: Build it, and they will come. Make something great and put it online. That is a big change from the recipe for winning in the past century.

4. Youth

The aspect of the Internet Bubble that the press seemed most taken with was the youth of some of the startup founders. This too is a trend that will last. There is a huge standard deviation among 26 year olds. Some are fit only

for entry level jobs, but others are ready to rule the world if they can find someone to handle the paperwork for them.

A 26 year old may not be very good at managing people or dealing with the SEC. Those require experience. But those are also commodities, which can be handed off to some lieutenant. The most important quality in a CEO is his vision for the company's future. What will they build next? And in that department, there are 26 year olds who can compete with anyone.

In 1970 a company president meant someone in his fifties, at least. If he had technologists working for him, they were treated like a racing stable: prized, but not powerful. But as technology has grown more important, the power of nerds has grown to reflect it. Now it's not enough for a CEO to have someone smart he can ask about technical matters. Increasingly, he has to be that person himself.

As always, business has clung to old forms. VCs still seem to want to install a legitimate-looking talking head as the CEO. But increasingly the founders of the company are the real powers, and the grey-headed man installed by the VCs more like a music group's manager than a general.

5. Informality

In New York, the Bubble had dramatic consequences: suits went out of fashion. They made one seem old. So in 1998 powerful New York types were suddenly wearing open-necked shirts and khakis and oval wire-rimmed glasses, just like guys in Santa Clara.

The pendulum has swung back a bit, driven in part by a panicked reaction by the clothing industry. But I'm betting on the open-necked shirts. And this is not as frivolous a question as it might seem. Clothes are important, as all nerds can sense, though they may not realize it consciously.

If you're a nerd, you can understand how important clothes are by asking yourself how you'd feel about a company that made you wear a suit and tie to work. The idea sounds horrible, doesn't it? In fact, horrible far out of proportion to the mere discomfort of wearing such clothes. A company that made programmers wear suits would have something deeply wrong with it.

And what would be wrong would be that how one presented oneself counted more than the quality of one's ideas. That's the problem with formality. Dressing up is not so much bad in itself. The problem is the receptor it binds to: dressing up is inevitably a substitute for good ideas. It is no coincidence that technically inept business types are known as "suits."

Nerds don't just happen to dress informally. They do it too consistently. Consciously or not, they dress informally as a prophylactic measure against stupidity.

6. Nerds

Clothing is only the most visible battleground in the war against formality. Nerds tend to eschew formality of any sort. They're not impressed by one's job title, for example, or any of the other appurtenances of authority.

Indeed, that's practically the definition of a nerd. I found myself talking recently to someone from Hollywood who was planning a show about nerds. I thought it would be useful if I explained what a nerd was. What I came up with was: someone who doesn't expend any effort on marketing himself.

A nerd, in other words, is someone who concentrates on substance. So what's the connection between nerds and technology? Roughly that you can't fool mother nature. In technical matters, you have to get the right answers. If your software miscalculates the path of a space probe, you can't finesse your way out of trouble by saying that your code is patriotic, or avant-garde, or any of the other dodges people use in nontechnical fields.

And as technology becomes increasingly important in the economy, nerd culture is rising with it. Nerds are already a lot cooler than they were when I was a kid. When I was in college in the mid-1980s, "nerd" was still an insult. People who majored in computer science generally tried to conceal it. Now women ask me where they can meet nerds. (The answer that springs to mind is "Usenix," but that would be like drinking from a firehose.)

I have no illusions about why nerd culture is becoming more accepted. It's not because people are realizing that substance is more important than marketing. It's because the nerds are getting rich. But that is not going to change.

7. Options

What makes the nerds rich, usually, is stock options. Now there are moves afoot to make it harder for companies to grant options. To the extent there's some genuine accounting abuse going on, by all means correct it. But don't kill the golden goose. Equity is the fuel that drives technical innovation.

Options are a good idea because (a) they're fair, and (b) they work. Someone who goes to work for a company is (one hopes) adding to its value, and it's only fair to give them a share of it. And as a purely practical measure, people work a lot harder when they have options. I've seen that first hand.

The fact that a few crooks during the Bubble robbed their companies by granting themselves options doesn't mean options are a bad idea. During the railroad boom, some executives enriched themselves by selling watered stock—by issuing more shares than they said were outstanding. But that doesn't make common stock a bad idea. Crooks just use whatever means are available.

If there is a problem with options, it's that they reward slightly the wrong thing. Not surprisingly, people do what you pay them to. If you pay them by the hour, they'll work a lot of hours. If you pay them by the volume of work done, they'll get a lot of work done (but only as you defined work). And if you pay them to raise the stock price, which is what options amount to, they'll raise the stock price.

But that's not quite what you want. What you want is to increase the actual value of the company, not its market cap. Over time the two inevitably meet, but not always as quickly as options vest. Which means options tempt employees, if only unconsciously, to "pump and dump"—to do things that will make the company seem valuable. I found that when I was at Yahoo, I couldn't help thinking, "how will this sound to investors?" when I should have been thinking "is this a good idea?"

So maybe the standard option deal needs to be tweaked slightly. Maybe options should be replaced with something tied more directly to earnings. It's still early days.

8. Startups

What made the options valuable, for the most part, is that they were options on the stock of startups. Startups were not of course a creation of the Bubble, but they were more visible during the Bubble than ever before.

One thing most people did learn about for the first time during the Bubble was the startup created with the intention of selling it. Originally a startup meant a small company that hoped to grow into a big one. But increasingly startups are evolving into a vehicle for developing technology on spec.

As I wrote in Hackers & Painters, employees seem to be most productive when they're paid in proportion to the wealth they generate. And the advantage of a startup—indeed, almost its raison d'etre—is that it offers something otherwise impossible to obtain: a way of measuring that.

In many businesses, it just makes more sense for companies to get technology by buying startups rather than developing it in house. You pay more, but there is less risk, and risk is what big companies don't want. It makes the guys developing the technology more accountable, because they only get paid if they build the winner. And you end up with better technology, created faster, because things are made in the innovative atmosphere of startups instead of the bureaucratic atmosphere of big companies.

Our startup, Viaweb, was built to be sold. We were open with investors about that from the start. And we were careful to create something that could slot easily into a larger company. That is the pattern for the future.

9. California

The Bubble was a California phenomenon. When I showed up in Silicon Valley in 1998, I felt like an immigrant from Eastern Europe arriving in America in 1900. Everyone was so cheerful and healthy and rich. It seemed a new and improved world.

The press, ever eager to exaggerate small trends, now gives one the impression that Silicon Valley is a ghost town. Not at all. When I drive down 101 from the airport, I still feel a buzz of energy, as if there were a giant transformer nearby. Real estate is still more expensive than just about anywhere else in the country. The people still look healthy, and the weather is still fabulous. The future is there. (I say "there" because I moved back to the East Coast after Yahoo. I still wonder if this was a smart idea.)

What makes the Bay Area superior is the attitude of the people. I notice that when I come home to Boston. The first thing I see when I walk out of the airline terminal is the fat, grumpy guy in charge of the taxi line. I brace myself for rudeness: remember, you're back on the East Coast now.

The atmosphere varies from city to city, and fragile organisms like startups are exceedingly sensitive to such variation. If it hadn't already been hijacked as a new euphemism for liberal, the word to describe the atmosphere in the Bay Area would be "progressive." People there are trying to build the future. Boston has MIT and Harvard, but it also has a lot of truculent, unionized employees like the police who recently held the Democratic National Convention for ransom, and a lot of people trying to be Thurston Howell. Two sides of an obsolete coin.

Silicon Valley may not be the next Paris or London, but it is at least the next Chicago. For the next fifty years, that's where new wealth will come from.

10. Productivity

During the Bubble, optimistic analysts used to justify high price to earnings ratios by saying that technology was going to increase productivity dramatically. They were wrong about the specific companies, but not so wrong about the underlying principle. I think one of the big trends we'll see in the coming century is a huge increase in productivity.

Or more precisely, a huge increase in variation in productivity. Technology is a lever. It doesn't add; it multiplies. If the present range of productivity is 0 to 100, introducing a multiple of 10 increases the range from 0 to 1000.

One upshot of which is that the companies of the future may be surprisingly small. I sometimes daydream about how big you could grow a company (in revenues) without ever having more than ten people. What would happen if you outsourced everything except product development? If you tried this experiment, I think you'd be surprised at how far you could get. As Fred Brooks pointed out, small groups are intrinsically more productive, because the internal friction in a group grows as the square of its size.

Till quite recently, running a major company meant managing an army of workers. Our standards about how many employees a company should have are still influenced by old patterns. Startups are perforce small, because they can't afford to hire a lot of people. But I think it's a big mistake for companies to loosen their belts as revenues increase. The question is not whether you can afford the extra salaries. Can you afford the loss in productivity that comes from making the company bigger?

The prospect of technological leverage will of course raise the specter of unemployment. I'm surprised people still worry about this. After centuries of supposedly job-killing innovations, the number of jobs is within ten percent of the number of people who want them. This can't be a coincidence. There must be some kind of balancing mechanism.

What's New

When one looks over these trends, is there any overall theme? There does seem to be: that in the coming century, good ideas will count for more. That 26 year olds with good ideas will increasingly have an edge over 50 year olds with powerful connections. That doing good work will matter more than dressing up—or advertising, which is the same thing for companies. That people will be rewarded a bit more in proportion to the value of what they create.

If so, this is good news indeed. Good ideas always tend to win eventually. The problem is, it can take a very long time. It took decades for relativity to be accepted, and the greater part of a century to establish that central planning didn't work. So even a small increase in the rate at which good ideas win would be a momentous change—big enough, probably, to justify a name like the "new economy."

Notes

[1] Actually it's hard to say now. As Jeremy Siegel points out, if the value of a stock is its future earnings, you can't tell if it was overvalued till you see what the earnings turn out to be. While certain famous Internet stocks were almost certainly overvalued in 1999, it is still hard to say for sure whether, e.g., the Nasdaq index was.

Siegel, Jeremy J. "What Is an Asset Price Bubble? An Operational Definition." European Financial Management, 9:1, 2003.

[2] The number of users comes from a 6/03 Nielsen study quoted on Google's site. (You'd think they'd have something more recent.) The revenue estimate is based on revenues of $1.35 billion for the first half of 2004, as reported in their IPO filing.

The High-Res Society

December 2008

For nearly all of history the success of a society was proportionate to its ability to assemble large and disciplined organizations. Those who bet on economies of scale generally won, which meant the largest organizations were the most successful ones.

Things have already changed so much that this is hard for us to believe, but till just a few decades ago the largest organizations tended to be the most progressive. An ambitious kid graduating from college in 1960 wanted to work in the huge, gleaming offices of Ford, or General Electric, or NASA. Small meant small-time. Small in 1960 didn't mean a cool little startup. It meant uncle Sid's shoe store.

When I grew up in the 1970s, the idea of the "corporate ladder" was still very much alive. The standard plan was to try to get into a good college, from which one would be drafted into some organization and then rise to positions of gradually increasing responsibility. The more ambitious merely hoped to climb the same ladder faster. [1]

But in the late twentieth century something changed. It turned out that economies of scale were not the only force at work. Particularly in technology, the increase in speed one could get from smaller groups started to trump the advantages of size.

The future turned out to be different from the one we were expecting in 1970. The domed cities and flying cars we expected have failed to materialize. But fortunately so have the jumpsuits with badges indicating our specialty and rank. Instead of being dominated by a few, giant tree-structured organizations, it's now looking like the economy of the future will be a fluid network of smaller, independent units.

It's not so much that large organizations stopped working. There's no evidence that famously successful organizations like the Roman army or the British East India Company were any less afflicted by protocol and politics than organizations of the same size today. But they were competing against opponents who couldn't change the rules on the fly by discovering new technology. Now it turns out the rule "large and disciplined organizations win" needs to have a qualification appended: "at games that change slowly." No one knew till change reached a sufficient speed.

Large organizations will start to do worse now, though, because for the first time in history they're no longer getting the best people. An ambitious kid graduating from college now doesn't want to work for a big company. They want to work for the hot startup that's rapidly growing into one. If they're really ambitious, they want to start it. [2]

This doesn't mean big companies will disappear. To say that startups will succeed implies that big companies will exist, because startups that succeed either become big companies or are acquired by them. [3] But large organizations will probably never again play the leading role they did up till the last quarter of the twentieth century.

It's kind of surprising that a trend that lasted so long would ever run out. How often does it happen that a rule works for thousands of years, then switches polarity?

The millennia-long run of bigger-is-better left us with a lot of traditions that are now obsolete, but extremely deeply rooted. Which means the ambitious can now do arbitrage on them. It will be very valuable to understand precisely which ideas to keep and which can now be discarded.

The place to look is where the spread of smallness began: in the world of startups.

There have always been occasional cases, particularly in the US, of ambitious people who grew the ladder under them instead of climbing it. But till recently this was an anomalous route that tended to be followed only by outsiders. It was no coincidence that the great industrialists of the nineteenth century had so little formal education. As huge as their companies eventually became, they were all essentially mechanics and shopkeepers at first. That was a social step no one with a college education would take if they could avoid it. Till the rise of technology startups, and in particular, Internet startups, it was very unusual for educated people to start their own businesses.

The eight men who left Shockley Semiconductor to found Fairchild Semiconductor, the original Silicon Valley startup, weren't even trying to start a company at first. They were just looking for a company willing to hire them as a group. Then one of their parents introduced them to a small investment bank that offered to find funding for them to start their own, so they did. But starting a company was an alien idea to them; it was something they backed into. [4]

Now I would guess that practically every Stanford or Berkeley undergrad who knows how to program has at least considered the idea of starting a startup. East Coast universities are not far behind, and British universities only a little behind them. This pattern suggests that attitudes at Stanford and Berkeley are not an anomaly, but a leading indicator. This is the way the world is going.

Of course, Internet startups are still only a fraction of the world's economy. Could a trend based on them be that powerful?

I think so. There's no reason to suppose there's any limit to the amount of work that could be done in this area. Like science, wealth seems to expand fractally. Steam power was a sliver of the British economy when Watt started working on it. But his work led to more work till that sliver had expanded into something bigger than the whole economy of which it had initially been a part.

The same thing could happen with the Internet. If Internet startups offer the best opportunity for ambitious people, then a lot of ambitious people will start them, and this bit of the economy will balloon in the usual fractal way.

Even if Internet-related applications only become a tenth of the world's economy, this component will set the tone for the rest. The most dynamic part of the economy always does, in everything from salaries to standards of dress. Not just because of its prestige, but because the principles underlying the most dynamic part of the economy tend to be ones that work.

For the future, the trend to bet on seems to be networks of small, autonomous groups whose performance is measured individually. And the societies that win will be the ones with the least impedance.

As with the original industrial revolution, some societies are going to be better at this than others. Within a generation of its birth in England, the Industrial Revolution had spread to continental Europe and North America. But it didn't spread everywhere. This new way of doing things could only take root in places that were prepared for it. It could only spread to places that already had a vigorous middle class.

There is a similar social component to the transformation that began in Silicon Valley in the 1960s. Two new kinds of techniques were developed there: techniques for building integrated circuits, and techniques for building a new type of company designed to grow fast by creating new technology. The techniques for building integrated circuits spread rapidly to other countries. But the techniques for building startups didn't. Fifty years later, startups are ubiquitous in Silicon Valley and common in a handful of other US cities, but they're still an anomaly in most of the world.

Part of the reason—possibly the main reason—that startups have not spread as broadly as the Industrial Revolution did is their social disruptiveness. Though it brought many social changes, the Industrial Revolution was not fighting the principle that bigger is better. Quite the opposite: the two dovetailed beautifully. The new industrial companies adapted the customs of existing large organizations like the military and the civil service, and the resulting hybrid worked well. "Captains of industry" issued orders to "armies of workers," and everyone knew what they were supposed to do.

Startups seem to go more against the grain, socially. It's hard for them to flourish in societies that value hierarchy and stability, just as it was hard for industrialization to flourish in societies ruled by people who stole at will from the merchant class. But there were already a handful of countries past that stage when the Industrial Revolution happened. There do not seem to be that many ready this time.

Notes

[1] One of the bizarre consequences of this model was that the usual way to make more money was to become a manager. This is one of the things startups fix.

[2] There are a lot of reasons American car companies have been doing so much worse than Japanese car companies, but at least one of them is a cause for optimism: American graduates have more options.

[3] It's possible that companies will one day be able to grow big in revenues without growing big in people, but we are not very far along that trend yet.

Upside, October 2001

In discussing the development of semiconductors in Silicon Valley, many roads originate with Arnold Beckman, the man who hired William Shockley away from Bell Labs and brought him to the San Francisco Bay area to establish the Shockley Semiconductor Labs of Beckman Instruments (now Beckman Coulter). Semiconductors had been around for several decades - odd materials that had the ability to conduct electricity under certain conditions. They are, Gordon Moore says, "halfway between insulators and metals. The wonderful thing about semiconductors is that you can control the amount of their connectivity through introducing impurities."

Bell Labs, as the research arm for the Bell companies - owners of millions of electromechanical relays across the nation used in its telephone switching networks - was one of the most interested parties in semiconductors and their ability to conduct electrical currents. Bell executives had the foresight to hope that, one day, Bell could replace its troublesome relays with more reliable devices made of semiconductors. Bell had also laid underwater cables that used vacuum tubes for repeaters at regular intervals, making the cables unreliable. So Bell funded a semiconductor laboratory in Murray Hill, N.J., which is where Shockley, Walter Brattain, and John Bardeen produced the work that received the Nobel Prize in 1956 "for their researches on semiconductors and their discovery of the transistor effect."

As the head of one of the country's leading scientific companies, Beckman understood the importance of semiconductors. Hence, he hired the greatest name in the industry to establish his company in that field, setting Shockley up in his own research and development facility. Although Beckman Instruments was based in Los Angeles, Shockley's new labs were set up near Palo Alto, CA, because Shockley's mother lived there.

The Shockley Semiconductor Labs were short-lived, however. With the exception of Robert Noyce, none of the key engineers working there could stomach Shockley for long, despite his unquestionable technical brilliance. In 1957, the labs' senior members selected Gordon Moore to contact Beckman and have Shockley moved aside as leader of the labs. When that didn't work, seven men - and eventually Noyce, making it eight - left the company. Moore still remembers the night he drove to Jean Hoerni's house to make the call to Beckman and also that it did little good.

In the end, Shockley may be most remembered for hiring the talented group and, some say, subsequently driving them away to join Fairchild Semiconductor. Moore, one of the "Traitorous Eight" who left to form Fairchild's semiconductor operation and eventually become one of Intel's founders, remembers the evening he was sitting at home in Maryland, when the phone rang, and the voice on the other end said, "This is Shockley." That's about all I remember about the call, but I took the job he offered. I had been doing pretty much esoteric work at Johns Hopkins University, looking at the spectroscopy of metals. I didn't know the first thing about semiconductors, but Shockley thought he needed a chemist. None of us knew his reputation as a manager at that time, but maybe we should have suspected, as none of his guys from Bell Labs were joining him in California." The son of a local policeman in the small coastal town of Pescadero, CA - directly west of what would become Silicon Valley - Moore wanted to return to California. Shockley's reputation and the incredible wage being offered, $750 a month, were all it took to bring Moore home, where he became one of the formative members of the group that would eventually make Santa Clara, CA, "Silicon Valley."

By the time Moore was on his way to California, commercial semiconductor manufacturing was underway in Boston, Phoenix, Dallas, and New Jersey. The establishment of the Shockley Semiconductor Labs was the first step toward adding the Bay area to that list. Many books have been written about the semiconductor industry's founding, with Charlie Sporck's book excerpted in our August issue as one of the few firsthand accounts (see "The Birth of Fairchild Semiconductor," August 2001, page 60). Trying to capture the semiconductor story for this issue of Upside meant tracking down the people who created the semiconductor industry - a nerve-wracking process. Noyce, eulogized in Upside (July 1990), was one of the first of his generation to pass away. But many of the other larger-than-life people from Fairchild Semiconductor - Eugene Kleiner, Jay Last, Pierre Lamond, Julius Blank, Andrew Grove, and Moore - are still around to provide insight into what it was like when the semiconductor industry was first being created. For this special issue, I particularly wanted to answer the question of whether such a technological discontinuity as the semiconductor revolution will ever appear again. The answer lies in a thorough understanding of how the Valley became "siliconized."

Just as Shockley knew the labs would need chemists, he knew that mechanical engineers would be required, so he hired two: Blank and Kleiner. Blank was a classic engineer and had worked at Babcock & Wilcox, where he designed and built the huge boilers used in power plants and utility companies. As a boy, Blank attended a technical high school in Brooklyn, where he learned the craft of building things. In 1943, the U.S. Army grabbed the young man, sent him to college, had him repair military aircraft, and then sent him to Europe to fight in World War II battles such as the Battle of Hurtgen Forest. By the time Blank returned to the States in 1946 to finish his bachelor's degree in mechanical engineering, he already had a lifetime's worth of practical experience. Then it was off to Babcock & Wilcox, Goodyear Aircraft, and, finally, Western Electric, which set him to work with germanium phototransistors, among other devices, to figure out how to replace its mechanical relays.

In 1956, Dean Knapic, a Western Electric alumnus, offered Blank a job at Shockley Semiconductor Labs. After traveling to Palo Alto to be interviewed by Shockley, Blank, like all the other original employees, underwent lengthy psychological testing - possibly an artifact of Arnold Beckman's experience in one of his firm's plants in Los Angeles, where an employee went berserk and stabbed a co-worker to death, or a result of Shockley's unorthodox views on personnel practices. Despite the days-long grilling, $10,000 a year plus moving expenses looked pretty good.

Blank and Kleiner shared all of the mechanical-engineering work at the company, which was housed in a small stucco building at 391 South Charleston Road in Mountain View, CA (now a chair shop that bears an incongruous brass plaque identifying it as the birthplace of Silicon Valley).

Blank's first assignment was to build a crystal grower. Blank knew nothing about semiconductors. Fortunately, Leo Valdez and Victor Jones, hired by Shockley to actually grow the crystals, shared what they knew about the type of equipment they needed, and Blank went to work. "That's what it was like then. Bobby Noyce would walk in and say, 'I want you to melt some copper on this part.' Really vague instructions were the order of the day. I would do that, and then take it in to him. He'd look at it and tell me how to change it, or make some other suggestions, and we would go back and forth like that, making things up," Blank recalls.

Despite the lack of direction, Blank loved the work. After he joined Shockley Semiconductor Labs, Noyce, Moore, Last, Kleiner, and Hoerni appeared in short order. Blank remembers the group as a bunch of 20-somethings who liked to hang out together and see each other socially. He remembers the entire year and a half at Shockley Semiconductor Labs as an exciting time, ordering power upgrades, phone systems, air conditioners, and the radio-

frequency (RF) oscillators needed to melt silicon. An indication of how Blank was regarded by his colleagues is the fact that Blank was nominated to recontact Beckman, after Moore's first attempt, about removing Shockley. "At first, it appeared I was successful," Blank says. Beckman endowed Shockley with a teaching chair at Stanford University that kept the good doctor out of the men's hair. Teaching, in addition to Shockley's speaking and travel demands after winning the Nobel Prize, initially seemed to have solved the issue. After a while, however, with Shockley coming back from trips and ordering entire projects restarted, it was clear that the problems would not be resolved. There was also Shockley's single-minded pursuit of the four-layer diode, perhaps left over from his days at Bell Labs, while many of the others thought silicon transistors were the direction they should be headed in.

Eugene Kleiner - co-founder, many years later, of the venture capital firm Kleiner Perkins - was another of Shockley's early hires. Kleiner was an immigrant from Europe. After leaving Vienna, Austria, he attended secondary school in New York. He then took work as a factory machinist, but, like so many of the remarkable men in this issue, he was drafted into the military. After the war, Kleiner earned a bachelor's degree from the Polytechnic University of New York and a master's in mechanical engineering from New York University. He taught engineering for a short while and then joined Western Electric, where he worked in the morass of the Bell system's relays and switches. Kleiner remembers Shockley as a charming person, a fascinating conversationalist, and, by reputation, one of the stars of Bell Labs.

Reconciling that impression of Shockley with the small, inadequate, and dirty building that Shockley had leased to start the company was the first of many events informing Kleiner that Shockley's technical brilliance was not matched by practical experience. Like Blank, Kleiner's first assignment was to build a crystal grower. It was this experience that brought Shockley down to earth in Kleiner's eyes. "I didn't know anything about growing crystals and [knew] nothing about furnaces. So I asked Shockley, but he didn't know. He just gave me advice, often the wrong advice, so our first device for growing crystals was a monstrosity. It didn't work. It was so large that we had to raise the roof of the building. It never worked. So I went next door to Sears, Roebuck and Co. and bought one of those large standing drill presses, and it had most of the features we needed. We had to change some things and add some things, but it formed the basis of our second and successful, crystal grower."

Then it was time to build a furnace, which entailed a similarly unsuccessful set of experiences. Still, "working with that team of men, Moore figured out the dispersal of gaseous materials," Kleiner recalls. (Moore was an expert glassblower. He actually created, by hand, the tube jungles in which gases were distributed, separated, and combined, thus allowing for single-step production of doubly diffused transistors.) "Knapic. Blank. It was a beautiful team. It was exciting," Kleiner continues. Once built, the furnace had to be kept on 24 hours a day. Kleiner and Blank, who lived close to the company, came in every couple of hours during weekends to check on the relatively delicate device. Kleiner remembers, 40 years later, that it was his wife who kept him going during those times (she also sang madrigals with Noyce at parties and social events). More importantly, she wrote a letter to Hayden, Stone & Co. seeking money for several of the "Fairchild Eight" - the letter that found its way to Arthur Rock and convinced Rock to find funding for Fairchild Semiconductor and, later, Teledyne Semiconductor and Intel. It was "one of the great letters of all time," Kleiner muses, thinking of that lovely unsung heroine.

Jay Last, now retired and living in Los Angeles, left Silicon Valley early on to become a vice president at Teledyne, where he helped Henry Singleton build that company from a single division to over 150 divisions. Last, a quiet electronics and optical engineer, worked closely with Noyce, and is responsible for much of the early manufacturing infrastructure of semiconductor technology. Surrounded by his collection of African art and the products of his publishing company, Last is, in many ways, uncomfortable sharing his life with strangers. His years of military

research and development have remained a closed book long after his retirement. And the subject of Shockley is one that, despite the passage of 40 years, is not casually remembered.

Still, if you catch his attention, Last will explain that much of what helped build the semiconductor industry came out of Fairchild Semiconductor: improvements in growing silicon ingots and diffusing exotic materials across the substrate of silicon wafers. Last's own contributions from optical science - the creation of photographic masks that were used to expose patterns on the substrates and the development of etching processes through which lines and connections were created on the wafers in miniature - are tossed off casually. "None of that had been done. We were inventing everything," Last says. This includes, most notably, adapting these processes to wafer fabrication and manufacturing. Waving his hands, Last dismisses the greatest revolution in manufacturing since Henry Ford's achievements, but it was Last's special contribution: "the creation of step-andrepeat methods by ganging up these microscopes and devices on mechanically operated stages, so that we could create integrated circuits cheaply, quickly, and reliably." He holds out one of the first commercially integrated circuits, with its five gold leads, showing me the four-transistor device with a sense of wonderment, even after all these years.

"And this," he says, retrieving yet another device from a trove of first technologies. I notice a tiny "open here" legend printed on its side. Last smiles conspiratorially and says, "We knew that the people at TI were desperate to see what we were doing, and we knew that, as soon as we released these, they would get some, so I had these printed up for them." He grins at his one-upmanship with boyish delight.

He won't say it, but those processes, step and repeat, led to "batch manufacturing," which spread out of semiconductor manufacturing and into other technological fields, such as LED manufacturing, biotechnology, and, experimentally, nanotechnology, fostering a worldwide change in manufacturing technologies that will continue throughout the 21st century. Last is sanguine, comfortable with his accomplishments and a hoard of unmentioned patents. I sit pondering this guarded, gentle, and brilliant man who tells me that, despite the passage of four decades, he can walk into a modern-day wafer-fabrication facility and still know what the machines do, which is basically what he and his colleagues first laid down in the 1950s. Last thinks back on Noyce, his closest collaborator at Fairchild Semiconductor, with a fondness that is touching - an almost universal emotion among the giants still walking the Earth who helped to create Silicon Valley.

While the men were in a quandary over their problems with Shockley, there were other companies working with semiconductors. Boston-based Transitron Electronic was a contemporary of the Shockley Semiconductor Labs. Transitron was found by an eccentric pair of brothers named David and Leo Bakalar. One of the pair ran a shoe factory and was cheaper than blue jeans; the other was a Ph.D. who had worked at Bell Labs on transistor research. They recruited engineers to their firm by traveling through Europe and conducting interviews in major cities. They hired the brightest immigrants they could find, brought them to the United States, and worked them silly in the early days of discrete electronics components. Pierre Lamond, Wilfred Corrigan, Robert Swanson, Lester Hogan, and a young Hungarian immigrant named Leslie Vadasz passed through Transitron's doors and built a semiconductor company that vied with Texas Instruments for the number one position in the nation in the late 1950s. At that time, semiconductors like germanium were employed to produce devices such as rectifiers in small aluminum canisters, which electronics manufacturers used by the thousands.

Pierre Lamond, a jeweler's son who fought in the Algerian War and made contacts with American officers during a stint at NATO headquarters, wanted to find a way to work in the United States. It was clear that his field (physics and

electron optics) was moving fastest in America. In 1957, a decommissioned Lamond saw an employment ad in the New York Times for Transitron. David Bakalar offered him a job on one of his recruiting trips through Paris. To get Lamond started, the Bakalars put him on the production line. In his third week, he was promoted to the head of production to replace his departing boss, and, in a few more months, he was promoted to device development engineer. It was a whirlwind introduction to semiconductor technology, and, for the one-year duration of his working visa, he absorbed everything that he could before returning to France. Lamond shopped for a company in his native country where he could apply his skills, but he found nothing comparable to his experience in Boston, so, by 1959, he was back at Transitron as the head of development.

Lamond's stint at Transitron was a short one, as most were. The Bakalars' approach was to "second source" parts designed by others (a polite way to describe their energetic copying of other people's products), which they would then sell to customers at cheaper prices. Lamond had heard of Hoerni's work at Fairchild Semiconductor in developing what would become known as the planar process of manufacturing semiconductor products. Lamond approached the Bakalars and told them that, if they were to compete, it was time for Transitron to invest in planar manufacturing, but they would have nothing of it, telling him to wait a few months until all the bugs were worked out of the process. Unfortunately for the Bakalars, the energetic Lamond had befriended Moore while presenting a paper at a technical conference, so the brothers' refusal to move Transitron along the path to original-product development came at a time when Fairchild Semiconductor's human resources manager was already courting Lamond.

In 1961, Lamond joined Fairchild Semiconductor, working for Moore. After one of the periodic diasporas of Fairchild's engineers to form a new company - in this case Signetics - and the loss of Lamond's direct supervisor, he was again promoted to lead device development. He remained at Fairchild from 1961 until 1967, when he became a member of another group leaving Fairchild, under Charlie Sporck, to restart National Semiconductor. Lamond remained with National until 1974.

Many of the Transitron staff passed through Phoenix (Motorola) or Dallas (Texas Instruments) before coming to Silicon Valley. Vadasz spent his three years of apprenticeship at Transitron, for $650 a month, before moving straight to Silicon Valley, where the man who had hired him into Transistron, Lamond, hired him again at Fairchild. As with others at Transitron, Vadasz found himself building whatever he needed. Need vacuum chambers to distribute evaporated dopants? Build them yourself, out of a piece of tube. Need some silicon wafers? Build your own furnace; dump in chunks of silicon lumps; start it spinning; dip in a piece of seed crystal, rotating it in opposite directions; pull it out slowly; and hope for the best. Everything else pretty much required working under a microscope with instruments held in your own shaking hands. Everybody who passed through the Boston sweatshop seems to have come out the better for having been there, and for having left.

For example, Wilfred Corrigan remembers someone from Transitron coming into a room, where he was reading about semiconductors, and telling him that the production line was a mess: "They told me to get out there and get it running right, so I went, and I figured out semiconductors later." Transitron was that kind of place.

Fairchild Semiconductor was, by comparison, a fortress of strong organization. Imagine coming into a company where at least a dozen future semiconductor company presidents and founders are working. At the head of the organization is Sherman Fairchild, founder of Fairchild Camera and Instrument, scion of one of the first investors in IBM, and head of a company with technology interests in multiple industries. A large part of the comparative stability

at Fairchild Semiconductor was based upon its hiring of Ed Baldwin as general manager of its new semiconductor division. Baldwin recalls, "I came up from Los Angeles, and they interviewed me - all of the them, Noyce, Moore, [Victor] Grinich, and Last - but it was Moore that made the decision to hire me."

Their choice was for someone as eclectic as themselves. Baldwin's father was an itinerant intellectual: a musician, engineer, and writer. His father was related to a former prime minister of England, where his family hailed from, and the founder of a technical university that still exists in London today. Baldwin earned one of Carnegie Mellon University's first master's degrees in solid-state physics and then a Ph.D. in nuclear physics, also from Carnegie Mellon, where he worked in neutron and proton scattering. Baldwin also built Carnegie Mellon's synchro-cyclotron, one of the progeny devices of the Berkeley cyclotron.

In 1950, the young scientist was recruited by Hughes Semiconductors, of Los Angeles, during that company's heyday. There, Baldwin worked with people such as Simon Ramo and Dean Woolridge, two of the founders of TRW, on the miniature diodes Hughes produced for airborne computers. By 1956, Baldwin had risen to the head of product development in Hughes' computer division, and, when that was folded, he became head of product development for the semiconductor division. While at Hughes, Baldwin wrote his first two patents for the company. The first patent was on semiconductor photodiodes, and the second patent was for a device that would use the diodes to read hundreds of mainframe punch cards per minute. The company promptly gave him a $100 bonus, patted him on the head, and sold his division to RCA for a small fortune. At that point, Baldwin was ready for a career change.

Baldwin remembers Hughes as a great training ground where he learned about operations, plant maintenance, engineering, manufacturing, personnel, and research and development. At Hughes, Baldwin also learned how to work with brilliant technologists and how to act as a professional manager. Nevertheless, an ad in the Wall Street Journal, seeking an executive and a general manager for a startup in Northern California, caught his eye. Baldwin had been looking for new work for a while, and, along the way, he'd befriended a banker who knew a source of capital for Baldwin's possible own company. But Baldwin hadn't heard from the banker in months, and the offer to work with the likes of Hoerni and Moore proved irresistible.

Baldwin had been at Fairchild Semiconductor for several months when the banker turned up, having secured a commitment from a wealthy industrialmanufacturing family by the name of Rheem. The banker was ready to press a check for $5 million into Baldwin's hands. "It was the worst decision of my life. If I hadn't made [that decision] I would probably be at Intel today," he says, looking wistful, and then he smiles and starts telling the story of a lifetime's worth of companies he has started and run. Baldwin reminds me that, at one point, an engineer who left Fairchild Semiconductor to join him at Rheem Semiconductor took one of Fairchild's precious "recipe books" - a source of considerable disgruntlement to this day. Baldwin recalls, "When I found out about it, I gave him the living daylights, and I returned the book." Today, at a spry 82 years of age, Baldwin is running a startup in San Diego, Energy Development Systems, where he will be working in his field "until the day I die," having just earned the most recent of several patents on the high-power capacitors that fuel his dreams for tomorrow.

While Baldwin was departing to found Rheem Semiconductor, Vadasz was in the midst of four years of work putting down deep roots into the development of bipolar, and later MOS, semiconductor products. But Vadasz was also destined to leave Fairchild Semiconductor, and was hired by Noyce and Moore to work in research at Intel, the company Noyce and Moore founded with the intent of designing and building semiconductor memories. There was a growing market for semiconductor components, but Noyce and Moore felt that, if they could build solid-state, or

thin-film, memories that would replace expensive and hard-tomanufacture solid-core memories, they would have ready customers among mainframe computer companies. It was a dubious market strategy at best, fraught with tough competition and only occasionally won by Intel. U.S. manufacturers ultimately abandoned the strategy and left memory making to Asian manufacturers.

Despite a well-published error having to do with a numbering scheme introduced much later, Noyce and Moore hired Grove and then Vadasz as their first Intel employees. Both Vadasz and Grove remain at Intel today. Ask Vadasz what he did in the beginning at Intel, and he'll explain that he worked on the technologies that led Intel into MOS manufacturing, the technique that is standard today. He goes on to admit that Intel's first successful memory products were good old-fashioned bipolar ones, like those of Fairchild Semiconductor. Press him for some details on the success he has brought Intel, and he'll tell you that it was Moore and others experimenting in the labs with MOS who finally figured out why the yields in the new technology were so low for Intel and made the process practical.

Out in the dry plains of Texas, Texas Instruments (TI) had been around since 1933, under another name, supporting the scientists working in the oil industry of Texas and beyond. A Texan giant of a man, Jack Kilby, had returned from Illinois to his home state with a bachelor's degree in electrical engineering to work with the resistors, capacitors, and discrete-logic products of his time basically to build the instruments that TI manufactured. The Iliac was just being built when Kilby graduated from college, but computing was firmly on the horizon, and Kilby increasingly turned his attention to germanium and then silicon, spending a lifetime in the semiconductor field, with a single company. Kilby can sit and preach the semiconductor bible from memory, because, at TI, he did what all of his competitors were doing elsewhere, slowly experimenting with semiconductor materials. He developed the design and manufacturing processes that allowed TI to "get there first," making TI the undisputed leader in transistor, and then semiconductor manufacturing.

There's a hard edge in the 90-year-old's clear blue eyes when you ask him about the industry back then. "We didn't share anything in those days. We were all competitors, in everything. We went to the ISSCC [International Solid State Circuits Conference] and announced our progress each year, but we said as little as possible beyond what we had done, and we certainly didn't tell anyone about our processes." Kilby, recently awarded a Nobel Prize for the invention of the integrated circuit at TI, doesn't talk much about his or TI's primacy in the field; he seems proud just to have been a part of the efforts in building one of the first multitransistor semiconductor microprocessors - a six-transistor part that TI built for use in products such as portable calculators and the Minuteman program. Kilby knows all the names from the Valley - Moore, Sporck, and others and has only praise to offer them and their work. Still, it seems as though his eyes burn bright, like fanned embers, when he remembers the competition of the time.

Awarding Kilby with the Nobel Prize for the invention of the integrated circuit is a touchy subject. No one has anything critical to say about Kilby, but a few people say that everyone knows that Noyce invented the integrated circuit. Still, the Nobel Prize - as was pointed out to me at least 10 times during the course of researching this article - is awarded only to living people; it's not awarded posthumously. So Kilby, who created an integrated circuit at about the same time as Noyce, ends up the sole winner. It's incongruous and a little bitter for everyone who loved Noyce, yet everyone is happy that the Nobel Prize committee finally recognized the achievement. It's one of those issues that makes people a little grumpy.

No account of the semiconductor industry would be complete without discussing Charlie Sporck, the creator, if not the founder, of National Semiconductor and the legendary head of manufacturing at Fairchild Semiconductor. He

retains the same gruff attitude he has always had, but a hint of deep humor resides somewhere under the attitude. Despite getting on in years, Sporck is about half a generation behind many of the people in this story. Sporck was classically trained engineer manager out of the General Electric system when he joined Fairchild Semiconductor as employee number 854. He was a senior executive at Fairchild Semiconductor when the famous Eight were well on their way to creating a new industry, and he left the company just in time to take over a tiny manufacturer of passive electronics components and bend it toward the innovation that he identifies as the mother of all inventions: the planar process, the basic manufacturing technique that defined an industry. In a nutshell, planar processes are the methods by which successive layers of semiconductors and conductors are deposited on a disc-shaped slice of a silicon ingot (and more exotic materials these days) and then successively coated, exposed, and etched with solvents, while keeping the junctions protected by silicon oxide.

Sporck's departure from Fairchild to National may have ratified the notion - tentatively introduced when the Traitorous Eight left Shockley - that it was OK to leave a lucrative career in a good company and move on because, simply put, there are better things to do with one's life and professional career than simply being a satisfied employee.

National Semiconductor latched onto planar techniques just as its arch rival, Raytheon, which had been content to slavishly copy National's products and designs for years, retreated from passive components and moved into what it thought would be greener valleys. National Semiconductor found its métier, pushing Raytheon (which had bought the declining Rheem Semiconductor operations) completely out of the semiconductor industry, and began a decade-long chase to extend the 16000 family of microprocessors. As is the way of the semiconductor, the 16000 family is also in history's dustbin, but rethinking that 20 years later is armchair quarterbacking.

In looking at this 70-year-old, still tree trunk-like and physically imposing, one thinks Sporck should have managed to push National Semiconductor over the top. He clearly had a feeling for the technology and was always at the head of industry wide issues. Sporck looked forward for the country as well as the industry. In retrospect, one wonders if it was the people or the lack of talent that prevented the company from beating Intel. Or perhaps it was just the sheer cussedness of Sporck's determination that somehow jinxed National Semiconductor as a contender. On the other hand, looking at National's new campus and growth, despite the microprocessor recession of today, it's clear that much of the infrastructure laid down under Sporck's reign still pushes the company along.

When Sporck left Fairchild Semiconductor, he took quite a group with him: Lamond; Fred Bialek; Floyd Kvamme; Don Valentine, who joined later; and Roger Smullen, who had joined Fairchild, as employee number 853, the same day as Sporck. Smullen started out at Fairchild as an entry-level engineer, fresh out of the University of Minnesota with a degree in mechanical engineering. Hired on at the princely salary of $540 per month, Smullen started in quality assurance, becoming a foreman in the wafer fab within eight months, under Lamond. Smullen was one of the first group of semiconductor engineers who attended the "Fairchild University" classes taught by Hoerni, Grove, Moore and Noyce. By studying under the men who had made up the techniques, Smullen learned how to do his job better, and he quickly moved up to manufacturing foreman and then to process engineer, under Sporck.

Smullen looks back on the Fairchild Semiconductor era and explains his and others' departures simply: "I got one share of stock as my reward for eight years of work, at an option price of $200 for the share. It was clear that these [Fairchild management] guys weren't going to share the wealth of what we were building." So, when Lamond asked

Smullen to come to National Semiconductor and run the digital circuit group, he left, spending four years at National, where there was a distribution of shares.

By 1971, Hoerni, in his third startup since leaving Fairchild, invited an increasingly well-known Smullen to run the digital memories division of the newly united Intersil and AMS companies, building RAM cards for IBM mainframes. Smullen worked with Jack Gifford, who ran the analog division of Intersil, before leaving Intersil in 1979 to take some time off. Smullen then joined Franklin "Pitch" Johnson in helping to revive Applied Micro Circuits Corporation (AMCC), a San Diego-based bipolar manufacturer that has since helped breath life into, and also benefited from, the return of bipolar technology for digital telephony. Clearly, Sporck had chosen well in hiring Smullen, and, just as clearly, Sporck had been right in abandoning Fairchild Semiconductor.

During the time between Sporck's departure from Fairchild to take over at National and Noyce and Moore leaving to found Intel, Fairchild found itself largely leaderless. It's too simple of a telling, but essentially true, that the manager of arch rival Motorola's semiconductor division, Lester Hogan, was chosen to run Fairchild; Darth Vader was invited into the company and shown all of its secrets. Rather than saving the company, many felt Hogan betrayed it. That was the downfall of Fairchild Semiconductor as the industry leader, and, although Fairchild is successful today, it is a largely reborn company, after two tortuous decades. Included in those two decades is yet another saga, National Semiconductor's acquisition and sale of Fairchild, but moving on to that story would force us to ignore the story of others at Fairchild, such as Wilfred Corrigan.

For a brief time, Corrigan, the eventual founder of LSI Logic, Jerry Sanders, the eventual founder of AMD, and others worked together under Hogan at Fairchild. Corrigan was a classically trained chemist, and chemists were greatly needed in the early days of semiconductors. He had worked in Boston at Transitron - the second-largest manufacturer of semiconductors, after Texas Instruments, at that time - before he and his new Norwegian bride jumped on a plane to Phoenix. Corrigan worked for Motorola for eight years, becoming a senior manager of Motorola's semiconductor operation, and then Hogan selected Corrigan to go to California, where he and the other "heroes," as they were called, would attempt to save Fairchild Semiconductor. Corrigan is one of the unsung technologists, and a much-recognized CEO, who helped a nascent industry develop the basic techniques, materials, and processes for semiconductor-wafer manufacturing. His area of specialization was in the use of RF microwaves, and similar technologies, to turn solid materials into gases and deposit them on ultra thin layers once again as solids, over the surface of photo masks that covered the wafers to produce the "integration of circuits."

Every semiconductor manufacturer in the late 1960s was fully integrated: grew its own silicon or geranium ingots, designed and manufactured its own circuits, performed its own testing, and integrated circuits into packaging suitable for their end use. Corrigan recalls that he couldn't even buy the exotic liquid mixtures that he needed for the processes he was attempting to develop and make precise. Instead, he had to gather the raw materials and mix them together, somehow managing to live through all of the explosions that occurred in his development labs, although his arms are permanently scarred from his work. After years of work at Motorola, the end result was the creation of repeatable techniques that reliably produced products of remarkably high quality. Hence, Corrigan's invitation to join Fairchild and his eventual uplifting to president of the semiconductor division, until that division was sold to Schlumberger.

It seems like hyperbole, but you look at Corrigan and think that he is a cross between Thomas Edison (Corrigan has several patents in his name) and John D. Rockefeller (Corrigan helped make an industry and now has the wealth to

demonstrate his record of accomplishments). From humble chemist to founder of a company that, 20 years later, has nearly $3 billion in revenue, he's still here just like his colleague Jerry Sanders. If a movie were to be made about Silicon Valley, the Hollywood studios simply wouldn't know what to do with a character like Corrigan: a scientist, entrepreneur, businessman, and marketer, with his great breakthrough at LSI Logic. He was the first to attempt, and succeed at, a commercial business based on building full-custom integrated circuits for others at a time when the industry had different models. Corrigan had the knowledge and the gonads to attempt to change the basic rules of operating in an industry created and populated by giants. Compare him to Edison, and he blushes and says, "Nonsense." Still, there's a Bentley in the parking lot, and, today, thousands of companies can afford to design and build application-specific integrated circuits largely as a result of one modest man who thought he could change the rules.

Another Transitron alumnus and transplant to Silicon Valley's semiconductor industry, Robert Swanson, founded a company the same year as Corrigan. But Swanson's interim learning ground was at National Semiconductor with Sporck. Today, Corrigan's company, LSI Logic, sits across the street from Swanson's Linear Technologies. Trained as an industrial engineer, Swanson found his way into Transitron because of his knowledge of statistics. Transitron was making linear-analog products as well as transistors, but it really had no way to track what was happening with different batches of materials that were cooked at different temperatures and produced varying quantities of usable products. Swanson's job was to figure all that out and improve yields. The statistics he knew cold; the rest this still-garrulous, twinkling-eyed Irish son of Boston figured out on the job. In the process, he became the manager of entire manufacturing lines, then multiple product lines, and then entire factories. By the time he went to work for Sporck at National Semiconductor, Swanson was starting up factories in Scotland and Germany, and he later ran the linear product line at headquarters.

"Charlie sued me for seven years after I left. I guess he figured that National was the only company that was supposed to be in the linear business. It was the biggest piece of business, and I did take some of the brightest National guys out with me," Swanson recalls. However, Sporck had done much the same to Fairchild Semiconductor before Moore and Noyce left. Swanson watched as National Semiconductor entered the watch business, minicomputers, and every new business on the block, while his own division, and the old-fashioned analog devices like power amplifiers or rectifiers that his division produced, got little of the limelight. He left and, 20 years later, runs a $2 billion business he founded producing the same kinds of parts that he once built for National Semiconductor. No bones about it: Swanson, the most youthful 50-year-old that you can imagine, built a global company in the face of disbelief, but with the support of VCs like Don Valentine, Thomas Perkins, and others who put up the $4.5 million Swanson and his partners wanted for 30 percent of their nascent company. In speaking with Swanson, you know that he's a guy you'd like to go drinking with; in fact, he looks like he might be pretty good in a bar fight, too.

Today, Linear Technologies builds the products in your personal computer that take the power delivered by the battery and separate it out into the individual power voltages required by the keyboard, the hard drive, the display, the input/output channels, and everything else. This is pretty complicated stuff that is still done best by the analog products that Linear Technologies sells around the world.

The final stellar graduate of the original semiconductor companies is the now more mature-looking T.J. Rodgers. In 1981, on the day after Linear Technologies' IPO, Rodgers' Cypress Semiconductor went public, with Morgan Stanley as the lead banker. Talk about a lack of coincidence; talk about a self-made man. Speaking of which, this man can talk. When he starts talking, after a while - when you get past the in-your-face presentation and the "I have an opinion on almost everything" personality - you hear the story of a young football player for Dartmouth College who

was too smart to remain a jock and who came to Stanford University to earn a master's degree and a Ph.D. in electrical engineering. Rodgers also remembers William Shockley, who was Rodgers' professor at Stanford the kind of professor who would invite his graduate students up to the blackboard to outline an answer to his question, let them write away for half a class, and then tell them how stupid they were. Tell Rodgers that he's a little like Shockley, and he won't disagree. He probably even likes the comparison in some ways. It's Shockley's brusqueness, forthrightness, intelligence, and impatience that he shares, and not Shockley's inability to lead or manage.

Rodgers was interested in a new form of MOS technology, VMOS, which promised to provide some significant advantages to semiconductor products. American Microsystems spotted Rodgers' dissertation, bought the patent on his VMOS idea, and hired him to boot. Rodgers spent the next five years discovering why VMOS was a dead end - lessons that he calls his MBA. The Rodgers went to work for Sanders at AMD. Thinking about these two men at one company presents some amusing images. Still, you can see that Rodgers would have contributed mightily to Sanders's success with AMD. But, like Corrigan and Swanson, Rodgers was destined to run his own show. Rodgers builds, as he proudly points out, static RAMs, the product business that Intel largely abandoned to the competition around 1984. If you could ever get this guy to shut up, you'd maybe tell him you think he's one of the smartest sons of a bitch you've ever met - and, in the context of this issue, that's saying something - but he's not about to give you the chance. Instead, Rodgers tells you he hates ass-kissers and is a reluctant republican, and then he shoves a copy of his latest ass-chewing memo to the company's engineers, regarding a recent mistake, under your nose.

We started this story about semiconductors with Moore, Noyce and Last, wondering about the odd, boyish enthusiasm that is still visible 30 years later. The story of Marcian Edward "Ted" Hoff and the making of the first microprocessor, the Intel 4004, is similar.

"Him, oh, he's some marketing guy," one person said to me, making me immediately think of William Davidow and preparing me to discover a young superachiever with a bachelor's, a master's, and a doctorate in electrical engineering, with an emphasis in what we have come to call artificial intelligence. Hoff was employee 12 at Intel, where he remained for what would be a wide and rich career for some, but was just the beginning for a self-admitted 60year-old inventor who keeps lasers and other technical wonders in his basement.

If Noyce was the grand old man, Moore the technical resource, and Vadasz the semiconductor wizard, then Hoff was the local postdoc, hired straight out of the labs at Stanford University, who was going to settle onto a workbench and make stuff happen. Hoff figured out how to push beyond the bipolar technology and products mandated by Noyce and into CMOS. And, by burying himself in the circuit lines and logic equivalencies of the semiconductors, he figured out that there were many different parts that Intel and other companies were designing and manufacturing that could be combined to make a device that served many generalpurpose functions. This device became known as the microprocessor and, more specifically, the Intel 4004 and then the Intel 8008 microprocessors. What kind of man is capable of pushing a company full of already-acclaimed geniuses into further achievements?

As you wander through a rambling conversation with Hoff, he is more interested in pointing out all of the individuals who did this or that work: how Federico Faggin's appearance at Intel made the microprocessor that he imagined possible and how the guy at the next workbench, Dov Frohman, invented the erasable programmable read-only memory, and introduced the concept of upgrading semiconductors once they left the factory. Bemoan the loss of the old Silicon Valley and the wonderful old places where you could push through dusty bins of spare parts and components - the kind of places that fired up a later generation of young inventors, like Steve Wozniak - and Hoff

puts you straight, rattling off a list of such places and their locations. He then tells you which place has what kind of parts before going into a diatribe about how the big argon laser that he bought at one of these places eats up a bunch of power and he can get one with what he is really interested in. At 65 years of age, Hoff is as much of a "Hardy Boy" today as he was at 20, and he probably will be until they put him in a box. Then we'll only have a few recollections of this kind of man, yet another of the special people who make Silicon Valley a land of dreams.

Hoff was managing a group at Intel when he hired a young Federico Faggin, straight from Italy with a doctorate in solid-state physics. According to Faggin, on his first day at Intel, an engineer from Japan Business Systems showed up and asked him about the product that he was building. When Faggin tried to explain that he was new to Intel, the Japanese representative flew into a tirade, telling him, "You bad," among other choice "jinglish" criticisms. After three days of tirades, Faggin convinced Masatoshi Shima, the Japanese representative, that he would design the custom circuit that the company had contracted with Intel to build, but he had to have a fair chance. So began a long-standing collaboration between Shima and Faggin, who would later hire Shima to work with him when he founded Zilog. Shima helped Faggin design the Z-80, the world's most produced microprocessor, which is still under license and being built today, but that's a story for another time.

When these two stood facing each other at Intel, Shima had the logic design for a chip that was to be included in a new calculator that Intel was supporting by replacing a handful of discrete-logic parts with a single chip. Faggin recalls that Noyce, Grove, and Vadasz were "all out worrying about memories. Intel was a memory company at that time, and the competition was thick. This full-custom thing was a side project for them; they were worrying about how to [recover] from a huge and recent failure in memories. This single-chip computer project wasn't even on their radar. They weren't concerned with it in the least. So I had to design and build the thing. Shima knew what functions he wanted in the calculator, and he had logic diagrams from the company. I had to figure out how to put that in silicon."

It's a long and convoluted story, but Faggin succeeded in building what was to become the first mass-production microprocessor at Intel. However, it was not until Japan Business Systems went into financial difficulties and released Intel from exclusive production limitations on the product that Faggin began to try to drag the Intel executives' attention toward what he had just created: the first microprocessor. "They just didn't get it," Faggin says. Well, they got it enough to allow Faggin to push ahead and design and develop the Intel 8008, a precursor to the current generation of microprocessors that is Intel's claim to fame, but it wasn't enough to keep a frustrated Faggin motivated. He left Intel, taking Shima and a young Ralph Ungermann (who later found the first Ethernet company) with him to Zilog, where they built the first microprocessor used widely in personal computers.

The semiconductor industry is rife with stories like this, and this is only a very truncated version of the tale. Still, if the Nobel Prize committee, in its wisdom, should decide to award a prize for a little piece of revolutionary silicon, Hoff - who, Faggin says, is the only person at Intel who supported his ideas and work, the only manager who "got it" - and Faggin would probably be named.

But, before we leave the semiconductor industry in the early '80s, let's spend a few moments with one other leader - a young man who grew up with 10 brothers and sisters in the cornfields of Indiana; attended the local technical university; graduated with a degree in electrical engineering; entered the U.S. Navy before the war had ended, but too late to fight; and started his first career at RCA. Bernard Vonderschmidtt was one of the early employees at RCA's solid-state division, during the company's heyday, and was elevated to run the division when it was making $500

million per year. Given his life and career thus far, building a hugely successful semiconductor division with a multinational company on the East Coast, Vonderschmidtt may not, at first glance, seem to belong in this pantheon of West Coast heroes. But allow me to explain.

Vonderschmidtt came to California in 1981 to run Zilog for Exxon, which had provided its founding capitalization. He stayed for three years before leaving, in disgust, as Exxon imploded in the computer field. One of the company's engineers, Ross Freeman, left with Vonderschmidt, and, together, they raised $4.25 million from John Doerr and several of Doerr's contacts. With that money, Vonderschmidtt and Freeman founded a new kind of semiconductor company called Xilinx, based upon the premise that the design of the microprocessor - or, more specifically, the logic employed by a designer in a chip's configuration - could be customized, as done with a full-custom chip (the freedom first given to designers by Corrigan and LSI Logic), but through the use of a common substrate of semiconductor product, more accurately referred to as a field-programmable gate array. In short, the designer takes logic-design software from Xilinx, applies a design to a base chip (the gate array), and programs the design onto a chip in a matter or days.

Similarly, the test function that often proves a full-custom design process has erred in some way is reduced to a matter of hours. In effect, the semiconductor revolution evolved into microprocessor, and, at Xilinx, the microprocessor has seemed to evolve to a stage where the customer, not the semiconductor company, determines to what use the silicon will be put. Today, Vonderschmidtt's company, which started with 600 gates (logic operations) on a chip, is about to produce gate arrays with 10 million transistors on a single chip. The company's four buildings, housing 2,500 employees, seem impervious to the current downturn faced by the general-purpose microprocessor-based companies, because the design of the microprocessor, according to Vonderschmidtt, is finally moving into the hands of those who will drive its future: the customers. It's an odd way to end this discussion of the microprocessor industry, thinking that the industry is about to change dramatically. But, after talking with the people in this story, you can well imagine that this is an industry likely to be reborn from within.